

# PYTHON PARA FINANZAS QUANT



**Primeros Scripts**

**t[0]**



 Juan Pablo Pisano  
@johnGalt\_is\_www

Impreso en:  
[laimprentadigital.com.ar](http://laimprentadigital.com.ar)

```
normalInv(x):  
    return ((1/math.sqrt(2*math.pi)) * math.exp(-x*x*0.5))
```

# PYTHON PARA FINANZAS QUANT

```
d1 = (math.log(S0/K) + (r - q + sigma*sigma*0.5)*T) / (sigma * math.sqrt(T))  
d2 = d1 - sigma * math.sqrt(T)  
ret['gamma'] = (normalInv(d1) * math.exp(-q*T)) / (S0 * sigma * math.sqrt(T))  
ret['vega'] = 0.01 * S0 * math.exp(-q*T) * normalInv(d1) * math.sqrt(T)  
ret['theta'] = (1/365) * (-((S0*sigma*math.exp(-q*T))/2*math.sqrt(T)) * normalInv(d1) -  
ret['rho'] = 0.01 * K * T * math.exp(-r*T) * fi(d2)  
except:  
    ret['errores'] = "Se ingresaron valores incorrectos"  
return ret
```

```
bsPut(S0, K, r, T, sigma, q=0):
```

```
ret = {}  
if (S0 > 0 and K > 0 and r >= 0 and T > 0 and sigma > 0):  
    d1 = (math.log(S0/K) + (r - q + sigma*sigma*0.5)*T) / (sigma * math.sqrt(T))  
    d2 = d1 - sigma * math.sqrt(T)  
    ret['put'] = K * math.exp(-r*T) * fi(d2) * math.exp(-q*T)  
    ret['delta'] = -math.exp(-q*T)  
    ret['gamma'] = math.exp(-q*T) * normalInv(d1) / (S0 * sigma * math.sqrt(T))  
    ret['vega'] = 0.01 * S0 * math.exp(-q*T) * normalInv(d1) * math.sqrt(T)  
    ret['theta'] = -0.01 * K * T * math.exp(-r*T) * fi(d2) * math.exp(-q*T)  
    ret['rho'] = -0.01 * K * T * math.exp(-r*T) * fi(d2)  
except:  
    ret['errores'] = "Se ingresaron valores incorrectos"  
return ret
```

## Primeros Scripts

t[0]

```
bsCall(S0, K, r, T, sigma, q=0):  
if (S0 > 0 and K > 0 and r >= 0 and T > 0):  
    maximasIteraciones = 300  
    tecno = prima  
    alfa = prima  
    i = 0  
    while i < maximasIteraciones:  
        i += 1  
        print(f'Iteration {i}/{maximasIteraciones}:')  
        print(f'Current price: {tecno}')  
        tecno = bsCall(S0, K, r, T, sigma, q)[0][0]
```

Created on Mon Mar 30 12:00:10 2020  
Author: Juan Pablo Toller UCEMA

math



Juan Pablo Pisano  
@JohnGalt\_is\_www

Impreso en:  
[laimprentadigital.com.ar](http://laimprentadigital.com.ar)

Introducción .....	11
Consideraciones generales .....	11
El Open Source .....	14
GitHub .....	15
StackOverFlow .....	16
Google .....	16
Lenguajes.....	16
La programación y la era de la Big Data .....	19
El futuro de la programación en el ámbito financiero .....	20
Las herramientas estadísticas y matemáticas .....	21
 Por que Python?.....	22
Las 8 premisas a evaluar en un programa o software .....	24
Diferentes entornos para programar .....	25
Instalaciones .....	26
Primer Hola Mundo en diferentes entornos .....	28
Consola básica de python .....	28
IDE con intérprete online .....	29
Spyder .....	29
Jupyter Notebooks .....	32
Editores de código, Atom, SublimeText .....	35
Ejercicios .....	36
 Variables .....	37
Asignación de valores a variables.....	37
Asignación de variables por consola o interfaz de usuario .....	38
Tipos de datos .....	39
Errores por operar tipos de datos diferentes .....	41
Python es un lenguaje interpretado, dinámicamente tipado .....	42
Operaciones Básicas .....	43
Métodos básicos para trabajar con strings .....	43
Palabras Reservadas.....	46
 Números al Azar .....	47
Valor flotante al azar entre 0 y 1 .....	47
Valor flotante entre min y max .....	47
Valor entre min y max definiendo intervalo o "step" .....	47

Número aleatorio en una Distribución.....	48
Operaciones Matemáticas. Librería Math.....	49
Constantes .....	49
Redondeo y truncado .....	49
Logaritmos .....	50
isClose .....	50
Ejercicios .....	51
Resuestas .....	52
Trabajando con Fechas.....	53
La librería datetime .....	53
Convertir un string en un Objeto de DateTime .....	55
Generar una fecha como objeto de datetime .....	56
Distintos formatos para mostrar fechas .....	56
Seteo de parametrización local .....	57
¿Qué es un timestamp? .....	59
Pasando a Otro huso horario.....	60
Jugando con Fechas y Calendarios.....	61
Ejercicios .....	64
Resuestas .....	64
Otras estructuras de datos en python.....	67
Listas .....	67
Slicing de una lista .....	69
Tuplas.....	73
Métodos de Tuplas y Listas .....	74
Métodos de las listas.....	75
Diccionarios .....	77
Inicialización de Listas y Diccionarios.....	80
Asignación múltiple.....	80
Ejercicios .....	81
Resuestas .....	83

Decisiones .....	87
Sentencia IF.....	90
Concepto de la indentación .....	90
Evaluación de True o False en distintos tipos de Variables .....	91
Decisiones concatenadas y anidadas .....	94
Decisiones Consecutivas .....	94
Decisiones Anidadas.....	95
Operadores Lógicos .....	96
Tablas de VERDAD .....	96
Manejo básico de errores, Try/Except.....	98
Catching, capturando el tipo de error .....	100
Ejercicios .....	101
Respuetas.....	104
Archivos necesarios .....	109
 Ciclos Finitos e Infinitos .....	109
Ciclos Definidos.....	109
Iterando una lista.....	110
Iterando una Tupla .....	110
Iterando una lista de tuplas .....	111
Iterando un Diccionario .....	111
Ciclos definidos usando la función range .....	111
Creando listas con ciclos definidos .....	112
Ciclos Indefinidos Finitos .....	113
Ciclos Infinitos.....	114
Corte de ciclos infinitos con corte por TimeOut.....	115
Corte de Ciclos infinitos con Centinela .....	116
Importación de datos de una planilla.....	117
Que es un archivo CSV .....	117
Librería CSV.....	117
Lectura de balances desde un CSV.....	123
 Bucles Anidados .....	125
Ejercicios .....	129
Respuetas .....	131
Títulos de esta colección .....	139

# Introducción

## Consideraciones generales

Si son personas pragmáticas que les gusta ir bien al grano directo en todo, salteen estas páginas introductorias que es todo sarasa, listo lo dije, al que quiera relajarse un rato y meterse en esta intro de a poco antes de ir al grano a continuación, mis palabras.

Esta obra nació de la idea de empezar a compilar material para una versión online o digital del curso preQuant que doy en la Ucema, me preguntaron infinidad de veces si pensaba hacer el curso en formato digital y lo primero que se me vino a la mente con esa pregunta es ¿cómo sería ese curso digital? ¿Qué recursos tendría aparte de unos vidéos mostrándome a mí codeando?, y no es un detalle menor, porque aprender las herramientas de análisis financiero cuantitativo requiere en realidad de la conjunción de 3 disciplinas: Programación, Finanzas y Matemática (Álgebra y Estadística, sobre todo).

Partí de la base que la mayoría de interesados tiene una base de conocimiento de instrumentos financieros, quizás sea porque me desempeño en este rubro y por eso veo ese background común entre los interesados, pero a su vez la gran mayoría tiene nulos conocimientos de programación y una base de matemática bastante pobre también, con lo cual el desafío es grande porque son todas disciplinas con una base teórica pero también con mucha base práctica. Siempre digo que aprender a programar es como aprender a manejar o a andar en bici, te pueden explicar la teoría en dos patadas y la verdad que no hay gran ciencia detrás pero de ahí a salir andando en bici por primera vez o a manejar en una ruta hay un gap que solo se resuelve con práctica no se necesita dedicar una vida ni mucho menos pero si muchas horas de práctica para dominar bien las herramientas de programación.

Volviendo al tema del nacimiento de la idea de esta serie de libros, empecé con la idea de armar guías de prácticas, sobre todo de las herramientas de python en forma gradual como las vamos viendo en el curso, y de a poco fue tomando forma de libro, el detalle es que son muchísimos temas y muchas herramientas y se hace medio denso y no iba a terminarlo jamás, así que decidí partirlo en tomos o capítulos, vamos a arrancar con el tomo 0, ya que en programación siempre se arrancan los índices con un 0, y si no fallan mis cálculos preliminares vamos a llegar al tomo 10 en los últimos temas en donde nos enfocamos en herramientas de machine learning y Deep learning o como lo llaman ahora que queda más cool aun "Inteligencia Artificial"

Los tomos siempre van a estar centrados en las herramientas informáticas en sí. Mientras escribo estas líneas ya tengo escritos los tres primeros tomos, el primero el T0 es un tomo de scripts iniciales, es decir las primeras líneas de código donde vemos variables flujo de un condicional y ciclos infinitos y finitos, etc, el tomo siguiente T1, se basa en una librería muy conocida de Python que se usa para trabajar con matrices o arreglos bidimensionales de datos (tablas de filas y columnas como las de Excel pero más estructuradas por decirlo de algún modo), luego el tercer tomo se basa en el uso de herramientas gráficas y el uso de funciones para no reescribir código, el cuarto tomo tengo pensado ya dedicarlo entero al uso de APIs y data feed, es decir a la recolección automática de datos en tiempo real en forma online y automatizada. Ya para el quinto tomo se empieza a poner más interesante porque con todas las herramientas vistas hasta ahí ya se pueden armar backtestings y screeners con buenos reportes gráficos. Luego vendrá seguramente un tomo entero dedicado

a bases de datos, es un tema central este para todo lo que es data science profesional y es un tema al que no se le da mucha bola en los cursos.

Y así la idea es como les dije antes terminar con algos de machine learning y siempre pero siempre todos los temas, la idea es que tengan toda ejercitación bien aplicada al campo financiero

## ¿A quién o qué público están apuntados los libros?

Me han preguntado mucho esto en mi cuenta de twitter, generalmente interesados en las herramientas mas cuantitativas y tecnológicas del campo financiero. Y es una pregunta difícil de responder, voy a separar el público objetivo en 3 tipos de interesados

1. Jóvenes, que trabajan en bancos, estudios contables, consultoría, o que se sienten atraídos por temas económicos y/o financieros, que quieren potenciar su desarrollo laboral.

Para estos jóvenes siempre el consejo es el mismo, más allá del tema financiero en sí, si estudiases abogacía o profesorado de educación física igualmente les diría que herramientas como python en 10 años van a ser como Excel hoy, y hoy mismo, saber python en sus carreras les va a dar un plus seguramente con lo cual ni lo dudaría en tomarlo como una capacitación más, si les gusta y se enganchan mejor, obviamente no es para cualquiera la programación pero si le ponen onda al menos a lo básico tampoco es algo con mayor dificultad

2. Profesionales de carreras de ciencias económicas, licenciatura en administración, carrera de actuaria, administradores de carteras de terceros o portfolio managers de fondos importantes. A todos ellos les digo que no se preocupen tanto por el tema técnico de python en sí, seguramente en su carrera laboral les tocará dirigir a jóvenes programadores que estudiaron a fondo este y otros muchos lenguajes de programación.

Pero también les digo que no pueden seguir capacitándose en otros temas dejando de lado esto por completo porque la tendencia al uso de big data, al uso de machine learning, al uso de herramientas cuantitativas que exploten cualquier mínimo sesgo estadístico para mejorar la eficiencia de un proceso es algo que llegó para quedarse. Yo soy el primero en estallar de la risa cuando escucho la preocupación de "¿nos remplazarán los bots?" porque es absurdo ese planteo, no no pueden, pero también es igual de absurdo creer que los bots son solo una moda y que podemos seguir trabajando en el futuro como lo hacíamos hace 10 años o lo seguimos haciendo ahora.. Ya está, les guste o no, la disruptión está instalada y falta la etapa de adaptación, competencia y mejora continua hasta la próxima disruptión, no tiene sentido resistirse

A todo este grupo de interesados, les diría que se lo tomen mas light, pero que le den bola, que entiendan que se puede hacer y que no, que demanda de trabajo y conocimientos conlleva, que velocidades y capacidades tienen estos sistemas, cuales son sus puntos débiles, y un sinfín de etc.

3. Por último hay un grupo no menor de interesados que son traders o inversores independientes, más allá de su trabajo profesional que puede no tener absolutamente nada que ver ni con las finanzas la economía ni incluso con los números pero que en sus finanzas personales, le dedican un tiempo importante a investigar cómo mejorar el rendimiento de sus inversiones personales. Hay de todo en esta selva, están los fundamentalistas del análisis técnico y los del análisis fundamental y también los que usan ambos tipos de análisis, también están los traders que hacen swings semanales o que operan con una frecuencia baja y los scalpers que hacen muchas operaciones diarias, y a su vez están los operadores de futuros, de acciones, de renta fija los loteros u operadores de opciones y dentro de estos los que son de armar estrategias sofisticadas, los arbitrajistas que adoran los sintéticos y los fanáticos que aman ir de frente a una posición especulativa, en fin, la selva es inmensa, pero les soy sincero, no se me ocurre un solo ejemplo al que este tipo de herramientas no le sume, en algunos casos como los más especulativos creo que el uso de este tipo de herramientas es casi un salvavidas que puede salvarles el pellejo más de una vez, en otros casos más de inversores del estilo "value" esto lo veo más como un complemento que les va a agilizar los screeners de activos y les dará un panorama más amplio de datos, comparadores y reportes pero a todos sin excepción este tipo de cosas les va a sumar un montón. Yo lo resumo así: Hoy operar/invertir sin python es como en los 90s hacerlo sin Excel.

## ¿Por qué vemos más de una vez lo mismo con herramientas diferentes?

¿por qué se dedica un capítulo entero a listas y diccionarios si después se dedica un libro entero a dataFrames de pandas?

Es algo que me pregunté antes de empezar a escribir la primera oración del libro ¿Qué herramientas muestro? Lo primero que se me vino a la mente es como programador más de base, mostrar las herramientas de base del lenguaje (por ejemplo, uso de listas y diccionarios) y después simplemente hacer una mención a Pandas y librerías super útiles como esas para trabajar con matrices de datos.

Pero después dije "no, al revés, si tengo herramientas tan completas como Pandas, para que los voy a torturar con listas llanas y recorrerlas así <peladas>, no tiene sentido" Entonces me autoconvencí de usar la mayor cantidad de librerías con el argumento que mejora mucho la curva de aprendizaje y hace la vida más fácil y la programación más entretenida.. peeeeero, ahí apareció el primer problema

¿Cuál es ese problema? El tema es que en programación por más copada que esté una herramienta (librería, paquete o lo que fuera que alguien pre-armó) siempre, pero siempre le va a faltar algo que queremos personalizar, y para eso vamos a tener que recurrir a la base <pelada> del lenguaje.

Es lo típico que nos pasa con cualquier solución "enlatada" por lo general están buenísimas para ese uso "enlatado" que pensó quien diseñó la solución, pero en donde necesitamos alguna cosita específica, ahí empezamos a decir "que lástima que no se puede hacer.." y si esas cositas específicas empiezan a ser muchas terminamos diciendo "esto solo sirve para..."

Así que si se preguntan en algún momento

*"¿para que me hizo dar tantas vueltas con esto, si después me enseñó, unos capítulos después, que se podía hacer mucho más sencillo con tal o cual paquete?"*

Bueno, en los párrafos precedentes está la respuesta. Es así, los paquetes y librerías están buenísimas, nos simplifican la vida, nos limpian el código, generalmente son más eficientes en uso de memoria y procesador para el uso típico, pero siempre tendrá alguna customización que no podremos hacer y tendremos que recurrir a "emparcharla" usando herramientas más de base, así que no queda otra que aprenderlas y si enseño solo las herramientas de base, me van a putear en todos los idiomas porque no les enseñé otras formas mucho más sencillas de resolver lo mismo con librerías

## El Open Source

Yo particularmente siempre fui muy desconfiado de las soluciones enlatadas, me daba cosa no saber como se resolvían las cosas "por detrás" por eso siempre fui a las bases, pero en python es todo "open source" ¿Qué significa esto?, que el código de base de todos los paquetes es abierto a la comunidad, es decir que cualquiera puede ver "la cocina" del paquete

Por ejemplo si usan una librería para calcular por ejemplo las primas teóricas de una opción según el modelo Black&Scholes o las griegas de las opciones, podrían decir "¿y yo que se si el que hizo esto consideró o no los dividendos o tal o cual cosa..?" bueno, esto no pasa en python porque al ser todo open source, si tienen dudas de cómo se calcula algo en una librería solo tienen que ver el código fuente de esa librería y está todo ahí, muchas veces hasta explicado con lujo de detalles, comentarios y ejemplos que puso el autor para mejorar la experiencia de los usuarios..

Me preguntarán "¿Dónde está la trampa de esto del open source?" y si, es lo primero que se nos viene a la cabeza, ¿por qué alguien haría algo gratuitamente abierto y lo compartiría con la comunidad dejando su trabajo así nomás expuesto?

Hay un sinfín de explicaciones, para un programador profesional las librerías que desarrolla son como la vidriera de un negocio, mientras más gente use sus librerías, más importante se considera su trabajo, así que para los desarrolladores profesionales esta puede ser una opción

Pero no es la única, muchos proyectos open source surgen de una bifurcación de proyectos privados donde el equipo de desarrollo tiene visiones diferentes de cómo mejorar el software, y allí por lo general la solución es separarse en dos proyectos y uno encara para un lado comercial no-open y el otro para el lado mas open source.

Y por supuesto que están también los casos mixtos donde un equipo de desarrolladores arma un paquete open source con la finalidad de darle una vuelta de tuerca en algún momento para una versión comercial o para vender el proyecto cuando tenga muchos usuarios a una empresa grande interesada por el product market fit del servicio o paquete, tal podría ser el caso por ejemplo de YahooFinance que en principio tuvo una API muy usada pero mucho, que luego quiso limitar, pero que en definitiva varios años después todavía

nadie sabe que es lo que quieren hacer, siguen manteniendo un feed de datos muy completo a nivel global y aun hoy no se decidieron si privatizar el proyecto o que.

Como sea, lo bueno de todas las librerías que seguramente veamos de python es que al ser open source pueden meterse en el código fuente y ver cada línea de código como está hecho cada cálculo, ver los argumentos que toma cada función y ver la documentación completa en el caso que la tenga.

Es más, no solo eso, los proyectos open source no solo permiten que la gente y usuarios sugieran cambios y/o mejoras sino que lo incentivan y agradecen.

Esto nos lleva a saber que en este mundo de los paquetes y librerías open source existen repositorios públicos de código y entre ellos el más conocido del mundo que es GitHub

## GitHub

Como dije antes GitHub es el repositorio de código más grande del mundo, pero no es solo un lugar donde se guardan y se pueden descargar los archivos (que además lo es, obvio), sino que los repositorios son además sitios dinámicos donde se guardan todas las versiones y cambios y documentaciones y además son participativos ya que permiten que varios desarrolladores mejoren el mismo repositorio al mismo tiempo e incluso permiten que cualquiera de nosotros hagamos una pequeña modificación en una "bifurcación del código" y les podamos sugerir a los dueños del repositorio que acepten nuestro cambio o sugerencia con solo un clic.. Es largo de explicar pero esto funciona de maravilla, se arman como "ramas" del código original en donde cada programador va haciendo modificaciones y esas ramas se van uniendo a la rama principal a medida que los dueños van aceptando los cambios y actualizaciones sugeridos.

Este tipo de repositorios dinámicos permiten que siempre esté online la versión más reciente y permiten además el trabajo colaborativo, lo que hace que estos proyectos sean monumentales a veces. Es muy loco, pero muchas veces uno usa una librería de estas y no se da una idea el inmenso trabajo que hay detrás y encima es todo gratis, con lo cual muchas veces la explicación a este lujo que nos podemos dar en estas épocas viene por el lado del trabajo colaborativo y la explosión de productividad que eso implica.

Me pasa mucho cuando le explico a alguien muy joven lo hermoso de esta época para aprender a programar comparado con lo que fue cuando aprendí yo, antes para lograr ver una tabla (ni hablar de un gráfico) de un análisis estadístico completo de una serie de datos temporales podría llegar a ser el trabajo de semanas de un programador de los años 90s, pero hoy quizás sean 10 líneas de código de un programador junior de python, sí, el mismo análisis.. esto es gracias a los paquetes open source, así que no sean quejoso y no se pongan a putear cuando algo no sale, valórenlo jaja

Bueno, no les digo más nada, averigüen el resto por su cuenta, vayan a [github.com](https://github.com) y busquen ahí, tienen repositorios de todos los lenguajes que se imaginen, entre ellos python y de todas las temáticas que se les ocurran, desde temas financieros, climáticos, de astronomía, deportes hasta fitness o arte.

## StackOverFlow

No podía hacer una intro a temas de programación sin mencionar [stackoverflow.com](http://stackoverflow.com) es el Google de los programadores, es un sitio que al principio medio que cuesta entender como funciona, es como una red social en realidad, hay usuarios expertos programadores en cada lenguaje y usuarios que solo preguntan y es como un inmenso foro de preguntas y respuestas de programación.

Lo bueno de stackoverflow es el formato e interfaz de usuario ya que los espacios para hacer preguntas y respuestas tienen un formato que permiten copiar y pegar código con formato de texto enriquecido para código, se van a dar cuenta lo que digo con la práctica, es muy útil porque a simple vista uno identifica muy rápidamente la respuesta que estaba buscando

Por otro lado otra gran ventaja es que permite que los usuarios califiquen las respuestas o veten por buena o mala digamos y esto le da un ranking a las respuestas haciendo que cuando uno busca algo que otro usuario ya preguntó (es lo que pasa siempre les aseguro) ve siempre primero las respuestas que mayor cantidad de usuarios calificó como buenas

Como les decía por lo general la duda de uno fue duda de otros en otro momento, solo es cuestión de buscar bien lo que uno necesita responder, a veces en forma de pregunta a veces simplemente copiando y pegando el error que nos tira el programa o la consola

## Google

De más está decir que la herramienta por excelencia es siempre Google, aunque en programación les diría que pelea cabeza a cabeza con stackoverflow, porque stackoverflow como les decía antes tiene un formato que lo hace muy piola para programadores y Google es más general, buscando en Google van a encontrar blogs, sitios oficiales de desarrolladores de paquetes, sitios de capacitación, en fin, de todo un poco, mientras quien en stackoverflow van a encontrar mas interacción entre programadores directamente.

No se dejen llevar por la frustración de que algo no les sale, me pasó mil veces eso, les aseguro, siempre hay días que uno se traba con algo y está 3 horas sin avanzar una sola línea, sobre todo al principio, pero insistan, googleen e insistan que siempre se encuentra una respuesta

## Lenguajes

Muchas veces me han preguntado por que lenguaje empezar a aprender a programar, y es una pregunta muy difícil, es como que te pregunten con que auto aprender a manejar.. y que se yo, si, no es lo mismo la caja de cambios de un fierro modelo 80 que de un auto modelo 2020 y ni que hablar comparado con uno de caja automática, hay muchas diferencias obviamente, pero sinceramente ¿conocen a alguien que haya aprendido con un determinado modelo y no haya podido manejar otro luego? Claro que al principio habrá una adaptación al nuevo auto, pero la base es la misma, luego de un tiempo de adaptación no hay que aprender a manejar de nuevo el otro auto sino adaptarse nomás..

Con la programación es lo mismo, si aprenden con Python que es un lenguaje de mas alto nivel que C++ si luego se pasan a C++ les va a costar mas que si aprenden con C++ y se pasan a Python, pero la base es siempre la misma, lo que hay que tener más en cuenta a la hora de elegir el primer lenguaje, es más bien el uso típico que le quieren dar, en el primer tema de este libro es lo primero que explico, pero sigo con la analogía con los autos.

Si quiero aprender a manejar porque necesito manejar una camioneta en el campo con caminos de ripio etc no me conviene elegir aprender a manejar con un Smart en Palermo en pleno tránsito y si necesito manejar para ir a ver clientes en medio de la capital todos los días no me conviene tanto practicar con un Falcon en las calles de tierra de la quinta de un amigo, ya que es mas práctico para ese uso, practicar con un auto mas parecido al que me vaya a comprar, en un lugar mas parecido al que me voy a mover.

O sea a lo que voy a la hora de elegir el auto (lenguaje) tiene mas sentido pensar en el uso que le voy a dar que en las características del auto (lenguaje) en si

## Aprender a aprender

Bueno, ya cerrando esta breve intro general al tema de programación, lesuento que todo lo que aprendan este año y el año que viene, en 10/15 años no va a servir mas, o algo así

Me dejo de joder un rato con el tema de los autos y les pongo el ejemplo con el famoso y querido Excel. Imagínense que hace 20/25 años hacían un super curso de "planillas de cálculo"

A principios de los 90s el Excel era toda una novedad, en épocas donde se usaban mucho el MsDOS y no tanto los entornos gráficos como Windows 3.11 que recién se empezaban a popularizar, en esa época las principales planillas de cálculo eran las del programa Lotus 1-2-3 que corría sobre MsDOS que no tenía entorno gráfico como windows, se veía como una consola y se usaban atajos de teclado difíciles de recordar, no tenían ni la mitad de las funciones que hoy tiene el Excel

The screenshot shows a vintage Microsoft Excel spreadsheet titled 'EMPLOYEE.DAT'. The columns are labeled A through F, and the rows are numbered 1 to 31. The data includes names like 'Allen', 'Brown', 'Jones', 'Smith', and 'Johnson', along with their salaries and years of service. The interface is in Spanish, with menu items like 'Archivo' (File), 'Editar' (Edit), 'Búsqueda' (Search), 'Formato' (Format), 'Filtros' (Filters), 'Ayuda' (Help), and 'Sistemas' (Systems). The font and overall design are characteristic of early 1990s software.

	A	B	C	D	E
1	ID	NAME	DEPT	SALARY	YEARS
2	10001	Allen	1000	\$4500	10.000
3	10002	Brown	1000	\$4500	10.000
4	10003	Jones	1000	\$4500	10.000
5	10004	Smith	1000	\$4500	10.000
6	10005	Johnson	1000	\$4500	10.000
7	10006	Doe	1000	\$4500	10.000
8	10007	Allen	1000	\$4500	10.000
9	10008	Brown	1000	\$4500	10.000
10	10009	Jones	1000	\$4500	10.000
11	10010	Smith	1000	\$4500	10.000
12	10011	Johnson	1000	\$4500	10.000
13	10012	Doe	1000	\$4500	10.000
14	10013	Allen	1000	\$4500	10.000
15	10014	Brown	1000	\$4500	10.000
16	10015	Jones	1000	\$4500	10.000
17	10016	Smith	1000	\$4500	10.000
18	10017	Johnson	1000	\$4500	10.000
19	10018	Doe	1000	\$4500	10.000
20	10019	Allen	1000	\$4500	10.000
21	10020	Brown	1000	\$4500	10.000
22	10021	Jones	1000	\$4500	10.000
23	10022	Smith	1000	\$4500	10.000
24	10023	Johnson	1000	\$4500	10.000
25	10024	Doe	1000	\$4500	10.000
26	10025	Allen	1000	\$4500	10.000
27	10026	Brown	1000	\$4500	10.000
28	10027	Jones	1000	\$4500	10.000
29	10028	Smith	1000	\$4500	10.000
30	10029	Johnson	1000	\$4500	10.000
31	10030	Doe	1000	\$4500	10.000

Ahora, si alguien aprendió a usar esas planillas de cálculo, seguramente se adaptó al Excel muy ráido porque aprendió a aprender.. y ¿a que me refiero entonces con aprender a aprender?

Me explico, seguramente muchos de ustedes hayan aprendido a usar Excel "probando" arrastrando fórmulas hasta que se dieron cuenta de usar el \$ para fijar una fila o columna o tipeando varias veces la misma fórmula hasta que descubrieron la función mágica "desref" o buscando valores a mano de otras tablas hasta que alguien les tiró el dato de usar "BuscarV" o "Index" y así.. o quizás hicieron un curso de Excel o quizás leyeron algún libro, pero la magia no está ni en el libro ni en el curso ni en nada de eso, la magia fue ir probando y las horas que estuvieron sentados frente a sus Excels probando para mejorar sus planillas.. Ojo lo mismo les pasó a los que aprendieron con el lotus 1.2.3 es decir lo valioso que aprendieron no fueron las herramientas en si, sino que "aprendieron a aprender" como funciona una planilla de cálculo.

Espero me hayan seguido el concepto, lo importante cuando vayamos viendo las herramientas de python no son las funciones que veamos sino digamos "la esencia de los algoritmos" y esa esencia es pensar de modo sistemico, es realmente dificil describir esto con palabras, seguramente a medida que avancen en este mundo de la programación me van a ir entendiendo mejor este concepto

Por ahora solo quiero dejarles esta idea de que se relajen, disfruten del camino y no se preocupen porque se pasaron un dia entero renegando con una sola linea de código, es muy común que pase eso, me pasó mil veces y me sigue pasando, pero cuando pasa eso no son horas perdidas para nada, porque en definitiva esa linea de código que no les salía es irrelevante, dentro de unos años no va a existir mas ese comando o quizás ni se use python ya porque lo remplazará otro lenguaje mas interesante vaya uno a saber, pero lo que si sé, es que esas horas que perdieron con esa maldita linea de código les enseñó a aprender y aunque no se hayan dado cuenta los fue forjando para pensar sistémicamente en "modo algorítmico" como suelo decir siempre.

## El futuro de la programación

Voy cerrando con una pequeña reflexión personal, hoy se estima que hay unos 20 millones de programadores en el mundo, cifra que era insignificante hace 30 años, es una carrera que no deja de crecer en el mundo, y que incluso tiene desempleo 0, ya hace años que la demanda crece mucho mas rápido de lo que las universidades forman a los nuevos profesionales lo que hace que muchos nunca lleguen a terminar sus carreras de grado, de hecho es casi inédito encontrar programadores con carreras de posgrado porque la demanda laboral es tan grande que hace irresistible para un estudiante de estas carreras abstraerse del mundo laboral para seguir capacitándose

Esto hizo que en los últimos 10 años surjan las famosas coding schools o bootcamps de programación que son como cursos muy prácticos e intensivos para formar programadores en tiempos records, ya que el mercado laboral los demanda como dije antes a mayor velocidad de lo que los forman las universidades tradicionales.

O sea que si pasamos de un número marginal de programadores en el mundo a 20 millones (de profesionales que viven de esto) en 30 años, imaginen ahora con el acceso a internet masivo, con la explosión de los cursos online, con la creciente demanda insatisfecha en casi todos los países del mundo de programadores, no es de extrañar que se proyecten unos 100 millones de programadores para 2025, 400 millones para 2030 y cerca de 1000 millones para 2035 o sea que en 15 años puede que más del 10% de la población haya adquirido capacitación en programación (no hablo de graduados universitarios)

A lo que voy con esto, imaginen en ese mundo de dentro de 15 años lo que va a influir herramientas como python en el trabajo cotidiano.

## La programación y la era de la Big Data

Implícitamente dentro de la programación hay un campo de trabajo que no para de tomar relevancia que es el campo de los "data scientists"

Voy a dedicar los siguientes párrafos para responder al final la pregunta de ¿qué demonios es un data scientist?, ya que este es uno de los principales campos de aplicación del lenguaje que veremos en esta obra, que es Python.

Si hay algo que crece a un ritmo super descontrolado son los datos, la cantidad de datos diarios generados por nuestra actividad cada vez que hacemos algo con nuestro celular o nuestra compu es increíble, cada clic o no-clic es un dato que se guarda en algún lado y muchas veces son datos valiosos si se saben cruzar e interrelacionar, pero ahí justamente está la clave, el dato suelto no sirve, ahora si te dicen que ese clic:

*"es un clic de fulano en un anuncio de tal marca que compite con tal otra en tal segmento de población y que el usuario tiene determinado perfil e hizo clic después de ver 4 anuncios de X competidores en los que no hizo clic en ninguno"*

La realidad es que tampoco nadie guarda los datos de esa forma, sino en bases de datos ordenadas y organizadas de tal forma que luego se puedan extraer para armar gráficos de correlación que permitan armar ese tipo de relatos de "fulanos genéricos", y digo "fulanos" a propósito, dirigido a los conspiranoicos que siempre están pensando que Google les quiere leer la mente.

En realidad la big data no son datos nominados, no pasa por ahí su valor, imaginen que hoy solo la parte de internet que se ve, alberga más de 15 zettabytes, los zettabytes son mil hexabytes que a su vez son mil petabytes que a su vez son mil terabytes que a su vez son mil gibabytes, un disco rígido de su compu lleno de información puede tener 100 gibabytes de data pura si hablamos de fotos o videos, pero si son solo datos numéricos, booleanos (Verdadero o Falso) o de texto, raramente supere los 2 gigabytes toda la info que uno guarda. Un Zettabyte son  $10^{21}$  bytes, es decir es un número con 21 ceros!

No parece tan grande? Veamos, supongamos que cada byte es un dato y que cada uno vale un dólar, en ese caso la fortuna de Jeff Bezos 100mil millones de dólares equivalen a 11 ceros, serían como 100 gigabytes como su disco rígido lleno de datos. El Sp500 todas las empresas juntas valen 30 millones de millones de dólares un número de 13 ceros el equivalente en bytes a 30 Terabytes

O sea que si con esta comparación de un dato un dólar, quisieramos alcanzar el valor de los datos de la internet visible necesitaríamos 500.000 veces las empresas del sp500, y les doy un dato mas, la internet visible es solo el 0,03% de la cantidad de datos de toda la internet profunda +privada +oculta.

Y pensemos una cosita más, esta cantidad infernal de datos se empezó a generar hace relativamente poco, piensen que Google existe hace solo 20 años, Facebook hace 15, es decir, en los próximos 15 años la cantidad de datos que generaremos será muchísimo mayor aún.

Volviendo al hilo de lo que quería decir, es irrelevante la comparación de 1 byte = 1 dólar obviamente, solo fue a modo didáctico y justamente lo que quería resaltar es la inmensidad del volumen de datos, pero ahí viene la siguiente pregunta ¿Qué vale mas, los datos o el análisis de los mismos?

Ese análisis es lo que transforma los datos en información, y la calidad de ese análisis lo que transforma esa información en información útil. Y a eso apunta la función de los data scientists.

## El futuro de los programadores

Digo todo esto, no como un incentivo a estudiar este tipo de carrera desde el punto de vista de perspectiva laboral, bien podría hacerlo y tengo infinidad de motivos, pero no es el punto al que quiero ir, lo que quiero remarcar es que todos estos programadores y científicos de datos, están dispersos en las áreas más heterogéneas que se pueden imaginar, desde la ciencia, el arte, el deporte, el ámbito administrativo la gestión pública, la banca, la industria logística, la salud, la manufactura, el ámbito educativo, los encuestadores, consultores, el marketing y la industria creativa, cinematográfica los videojuegos la industria del entretenimiento y podría dedicarle páginas enteras a la descripción.

¿Y a que voy con esto? A que hace 30 años todas esas industrias funcionaban perfectamente sin programadores ni científicos de datos agregando valor a sus actividades, y ahora ya ninguna es competitiva sin ellos.

¿Y? y que claramente la tendencia no se detiene pero recién estamos en muchos países en una etapa de "punto de inflexión" esos puntos en donde cambia la pendiente de la curva,

Lo que quiero decir que hasta ahora (+/- un par de años) dependiendo la industria, muchas empresas no se adaptaron aún a la disrupción tecnológica que significa la era de la big data, o recién están empezando tibiamente a dar los primeros pasos, pero dentro de muy poco cambiará muy rápido el escenario y con ello la forma de tomar las decisiones o de implementarlas o testearlas antes. Y hay que estar preparado para ello cada uno desde su lugar, no importa cual fuera, mientras mayor grado de responsabilidades mayor es la urgencia en entender esta disrupción mas allá de la técnica y las herramientas en si, y a menores escalas en la toma de decisiones mayor urgencia en entender las herramientas en si y aceptar la disrupción.

## El futuro de la programación en el ámbito financiero

En nuestro pequeño mundillo de las finanzas el tema del análisis de datos, la big data, python, c++, machine learning y la "sea in car" (expresión tuitera de hace unos días q me hizo mucha gracia) es un hecho consumado. Quiero decir, es una de las actividades pioneras en adoptar esta disrupción

Y no hablo solo del "trading" y los bancos de inversión como muchos se imaginan, el HFT y todo ese tema, sino que todo este rollo ya está presente en infinidad de ámbitos de la industria financiera, tanto en la banca, como en los seguros, y ni hablar en el mundo de las Fintech que vienen a patear el tablero con la competitividad que les da esta disrupción para desplazar a la banca tradicional en infinidad de rubros.

Bueno, doy por cerrada esta rápida introducción al mundo de los algoritmos, créanme que el viaje vale la pena, y si llegan a la conclusión que no lo vale es porque no es lo suyo y todo bien, pero seguramente les

sume un montón porque ya nada va a volver a hacerse como se hacía antes de que estas herramientas sean tan masivas y populares.

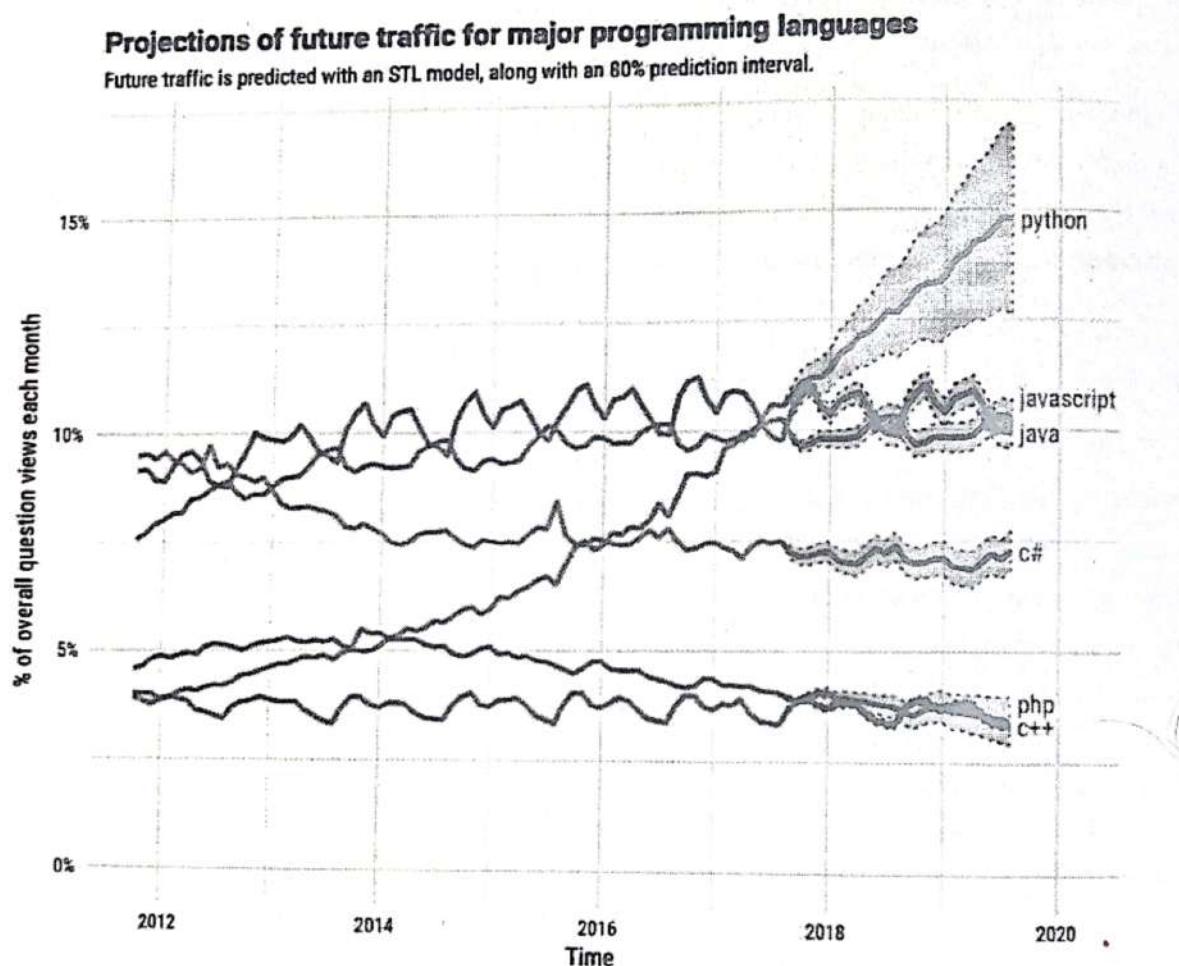
## Las herramientas estadísticas y matemáticas

Si bien esta serie de libros son libros de programación para aprender a programar, son necesarios, para muchas de las cosas puntuales de análisis cuantitativo que veremos, algunos conocimientos previos de análisis matemáticos, estadística descriptiva, álgebra, etc. Pensé en hacer un libro introductorio a estos temas como nivelación, pero preferí ir mechando de a poco a medida que vamos avanzando en la profundidad de los análisis este tipo de conocimientos.

Me contuve de no poner mucha fórmula ni nada que no sea estrictamente necesario para presentar el tema, así que dejo solo el comentario aquí mencionado que luego del tomo 10 calculo, vendrá un libro específico donde me meta mas en profundidad en temas matemáticos y de modelización para los que están mas interesados en esa rama del análisis cuantitativo.

## Por que Python?

En términos generales está de moda, es decir de todos los lenguajes para todos los usos, es el lenguaje del momento, esto quiere decir que si armamos algo, vamos a poder compartirlo con más gente, y a su vez vamos a poder ver más cosas compartidas por otros desarrolladores si usamos este lenguaje, esto incluye librerías, SDKs, ejemplos, tutoriales, ayuda en solución de problemas, lo que en el ambiente llamamos "la comunidad"

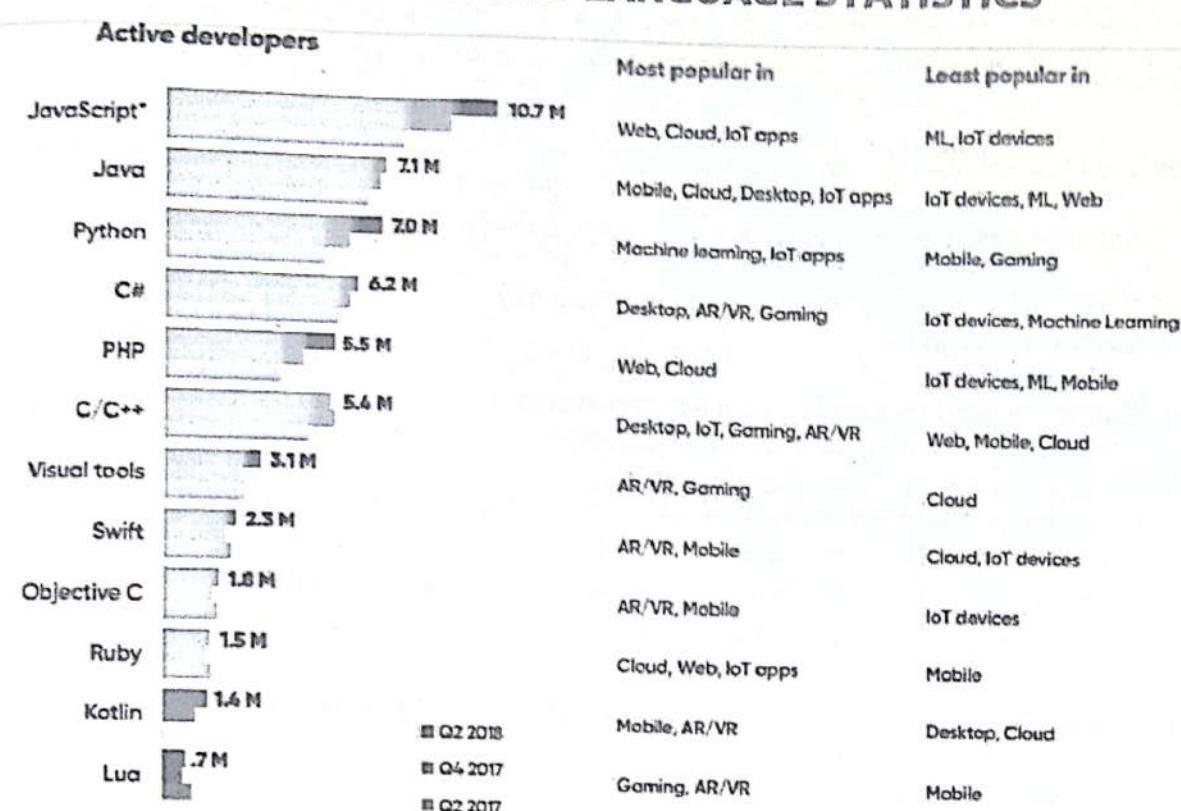


Por otro lado cada lenguaje tiene su uso más típico o digamos para lo que es ideal y usos para lo que no conviene ni pensarlo

Es como todo cada cosa tiene sus pro y sus contras y eso lo hace mas o menos útil para determinados usos específicos

En el cuadro de la página siguiente les muestro un resumen muy interesante publicado en el sitio stackoverflow.com (uno de los sitios mas consultados por la comunidad mundial de programadores) en base a su encuesta anual de lenguajes de programación

## WORLDWIDE PROGRAMMING LANGUAGE STATISTICS



- Ejemplo, el lenguaje **C** es un lenguaje de bajo nivel, es decir, un lenguaje más cercano al hardware, por eso se los llama lenguaje de máquina, por lo tanto es utilizado en proyectos en los que se requiera más cercanía entre el código y el hardware, generalmente proyectos en donde la performance es vital (Por ejemplo HFT)
- Ojo que **C++** no es lo mismo que **C**, es una evolución del mismo, a su vez **C#** es una evolución de **C++** desarrollada por Microsoft, y que compite con **Java** o sea que es de mucho más alto nivel
- Otro ejemplo de lenguaje muy utilizado es **Java** que es compilado, es el referente en programación orientada a objetos y muy utilizado para interactuar con todo tipo de dispositivos, es muy usado para apps nativas para mobile por ejemplo o para aplicaciones de escritorio.
- En cambio, **PHP** por ejemplo es el lenguaje del lado del servidor por excelencia para páginas y aplicaciones web, hoy en día se dice que el 80% de la web está hecha en **PHP**
- Sin embargo, **Javascript** está ganando terreno gracias a **NodeJs** que permite utilizar este lenguaje para el backend (del lado del servidor)
- Python**, es un lenguaje de muy alto nivel, es decir muy fácil para escribir código por la gran cantidad de librerías y código prescrito que tienen, su uso más típico es para Inteligencia artificial, machine learning, data science (en donde compite con **R**) pero también se usa para aplicaciones web en menor medida y es un lenguaje muy versátil y con una curva de aprendizaje muy amigable con lo cual es genial para aprender

De todos modos hay infinidad de lenguajes, todos tienen alguna ventaja y algún uso para los que son geniales y otros para los que no son lo ideal, la verdad que para aplicaciones financieras, backtesting, screenings, recolección de datos, etc es muy recomendado mas aun teniendo en cuenta que consta con una comunidad que no deja de crecer

# Las 8 premisas a evaluar en un programa o software

Antes que nada tiro unos tips muy importantes:

- Siempre pero siempre hay una solución al problema planteado
- Siempre que hay una solución a un algoritmo hay muchas maneras de llegar al mismo resultado
- Cuando el programa no funciona no es nunca la máquina, mucho menos una conspiración mundial, somos nosotros que pensamos mal la lógica

Dicho esto, como siempre hay muchas maneras de llegar al mismo resultado, es decir que no hay una solución única, lo importante en programación no va a ser llegar a la solución correcta del problema, sino llegar de la forma mas razonable posible dependiendo de las necesidades del caso, para exemplificar veamos las premisas con las que siempre vamos a lidiar

1. Confiabilidad: ¿Resuelve siempre sin fallas? ¿De qué depende? ¿Cuál es el % de fallas?
2. Testeabilidad: ¿Hay maneras de probar el programa antes de sacarlo a producción?
3. Performance: ¿Se podría hacer que sea más rápido ejecutándose o que use menos recursos de memoria/procesador?
4. Usabilidad: ¿Es fácil de usar para un nuevo usuario o para mi mismo?
5. Mantenibilidad: ¿Que tan sencillo es realizarle cambios? ¿qué tan prolífico y legible queda el código?
6. Escalabilidad: ¿Como se comportaría si crece en cantidad de datos, de usuarios, de instrumentos?
7. Portabilidad: ¿Me sirve para otro mercado, instrumento, horario, etc?
8. Seguridad: ¿Está protegido de ataques?

Siempre, pero siempre habrá muchas maneras de resolver un problema, y ni hablar en programación, las variantes son infinitas diría, pero también siempre habrá limitaciones y prioridades, de esas 8 premisas nunca jamás se puede ser óptimo en todo, siempre que se elige una prioridad de las 8 se resigna otra (en cierto grado obviamente) por lo tanto es muy pero muy importante siempre pensar antes que nada cual va a ser la prioridad que vamos a tomar

Por ejemplo la **confiabilidad** va a ser la prioridad número 1 en un bot de **backtesting** de un método de trading, pero si quisiera un programa para empezar a **probar una idea**, probablemente no me interese tanto el fino de los datos y los resultados sino que pensaría primero en la **testeabilidad** porque para ser más riguroso con los datos hay tiempo luego en la etapa de producción pero en una primera etapa necesito ser más flexible a probar muchos cambios y soportar errores típicos de una primera etapa de prueba

Si estoy haciendo un bot de **screening** en tiempo real, donde la cantidad de datos es infernal y la velocidad de procesamiento es crucial, ahí la prioridad número 1 va a ser la performance ya que al ser un screening puedo dejar para un segundo procesamiento la responsabilidad de la confiabilidad

En fin, son solo ejemplos para mostrar que antes de la primera línea vamos a tener que tener este tipo de cosas presentes

## Diferentes entornos para programar

Podemos agrupar a las herramientas para escribir código en 5 grupos, de más básico a más completo y serían los siguientes:

- Consola de Python: Es la forma más rápida de probar un script, pero terriblemente incómoda, mas que nada se usa para ejecutar
- Jupyter notebooks: Vienen en Anaconda, son piolas para ir probando código línea por línea pero no sirve de mucho para un proyecto en si
- Intérpretes online: Ej `repl.it` es interesante para cuando no tenemos nuestra compu y queremos trabajar en proyectos chicos
- Editores de código: Serían como un Word pero para codear, los más recomendados son: **SublimeText, Atom**
- IDEs: Son entornos de desarrollo integrados (integran consola + editor de texto etc), ejemplos: **Spyder, Pycharm**

La principal ventaja de los **Jupyter Notebooks** es que combinan anotaciones con texto enriquecido y widgets con código lo que los hace muy piolas para lo didáctico, a su vez otra gran ventaja es que permiten ir ejecutando código línea a línea y reiniciar el kernel solo cuando haga falta

En cambio, en los **intérpretes online** la principal ventaja es que no tenemos que instalar nada, simplemente abrimos una web y listo y si nos registramos podemos dejar guardados nuestros proyectos en carpetas.

Obviamente no sirven mucho para grandes proyectos pero para tener algunas cositas siempre a mano vienen muy bien

Por el lado de los **editores de código**, a mi gusto es lo mejor para el programador profesional porque tienen todo tipo de facilitadores, atajos de teclado, funciones de autocompletado etc, lo que acelera la productividad mucho y son super livianos y rápidos

Por último, tenemos los **IDE** que son **entornos de desarrollo** completos, donde tenemos además del editor de código con todas sus funciones, una consola integrada y en algunos casos herramientas de debug más avanzadas para ir viendo el paso a paso de nuestra ejecución si lo quisieramos, suelen ser bastante más pesados que los editores de código, y enlentecen un poco la ejecución

Para este curso y sobre todo para principiantes recomendamos enfáticamente la instalación de Anaconda La principal ventaja es que con una sola instalación ya tenemos de una:

- Python 3.7
- Un prompt en la carpeta de instalación para instalar al toque las librerías externas
- Spyder que es un IDE muy completo y práctico
- jupyter Notebooks (cuadernos para codear paso a paso con anotaciones enriquecidas)
- Las librerías más usadas como pandas, numpy, matplotlib etc

## Instalaciones

Arrancamos, paso 0 instalación del software, en este punto yo recomiendo para principiantes instalar el paquete anaconda por varios motivos

- Tiene python incorporado
- Viene con las librerías más usadas que son pandas, matplotlib, scikitlearn, etc.
- Cuenta con un IDE, que es el spyder, muy práctico para principiantes
- Cuenta con Jupyter Notebook
- Viene con un prompt que funciona muy bien en todos los OS que sin necesidad de navegar por carpetas nos permite instalar librerías fácilmente

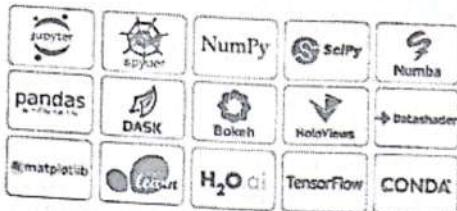
### Tips de instalación

- Descargarlo de [www.anaconda.com](http://www.anaconda.com)
- Instalar la versión con python 3.7
- No cambien la carpeta default de instalación
- Es recomendable que la ruta de instalación no tenga directorio con acentos, eñe u otro carácter no estándar
- Si lo instalan en una notebook con un firewall con muchas limitaciones pueden tener errores de instalación
- Requerimientos: Con una notebook cualquiera de 2Gb de RAM, medio pelo año 2018 en adelante alcanza perfecto

## Capturas de pantalla de la página oficial de anaconda

The open-source Anaconda Distribution is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 1.9 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling individual data scientists to:

- Quickly download 7500+ Python/R data science packages
- Manage libraries, dependencies, and environments with Conda
- Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
- Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews



Windows | macOS | Linux

### Anaconda 2019.10 for Windows Installer

#### Python 3.7 version



64 Bit Graphical Installer (462 MB)  
32 Bit Graphical Installer (410 MB)

#### Python 2.7 version



64 Bit Graphical Installer (413 MB)  
32-Bit Graphical Installer (356 MB)

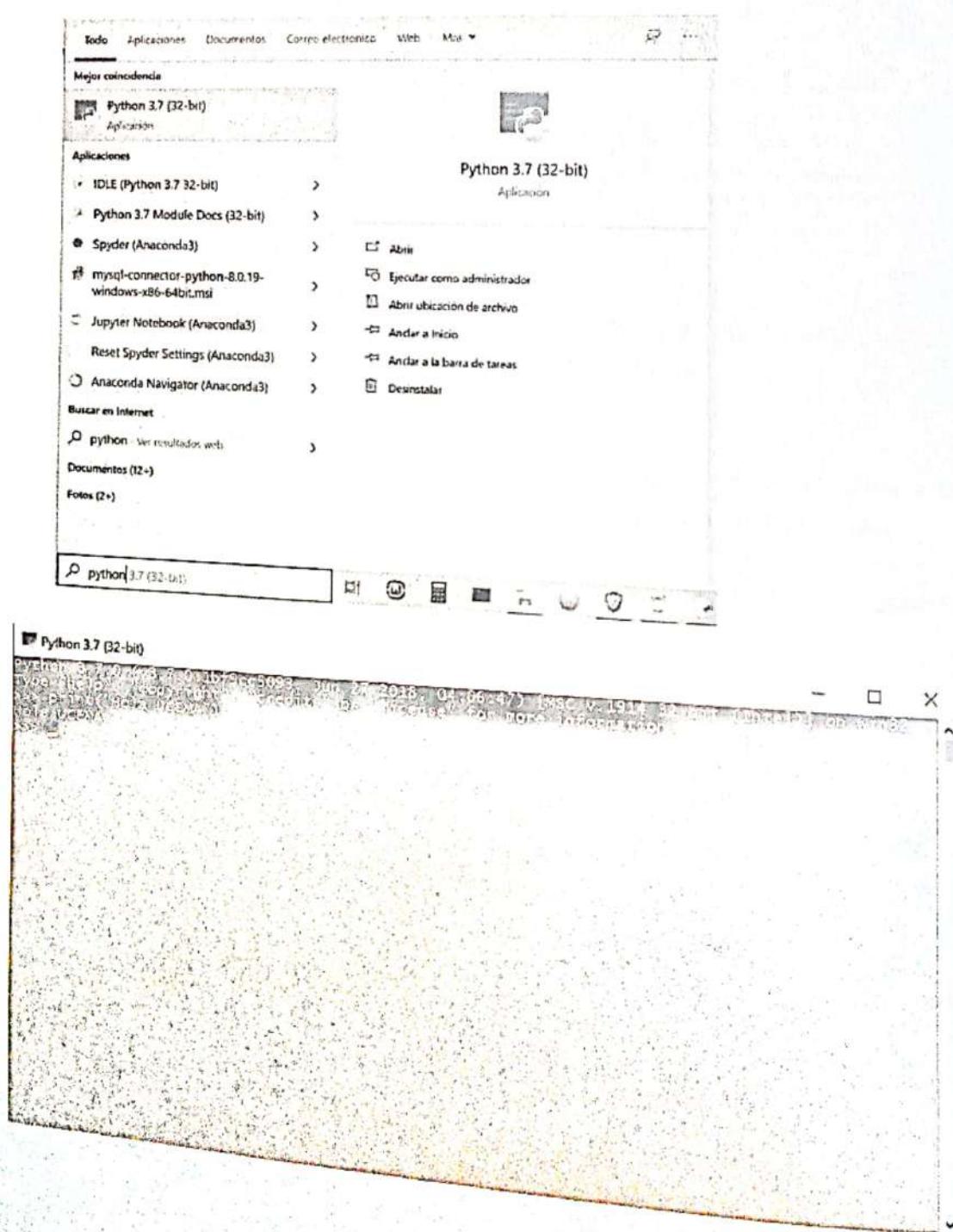
# Primer Hola Mundo en diferentes entornos

## Consola básica de python

Arranquemos con la consola pelada de python, buscamos python en el buscador de programas de windows y lo ejecutamos

Una vez abierto python ponemos la instrucción y le damos enter para ejecutarla

La verdad que va a ser muy raro que ejecuten código directo desde la consola de python pelada, yo lo he usado solo para hacer alguna prueba cada tanto

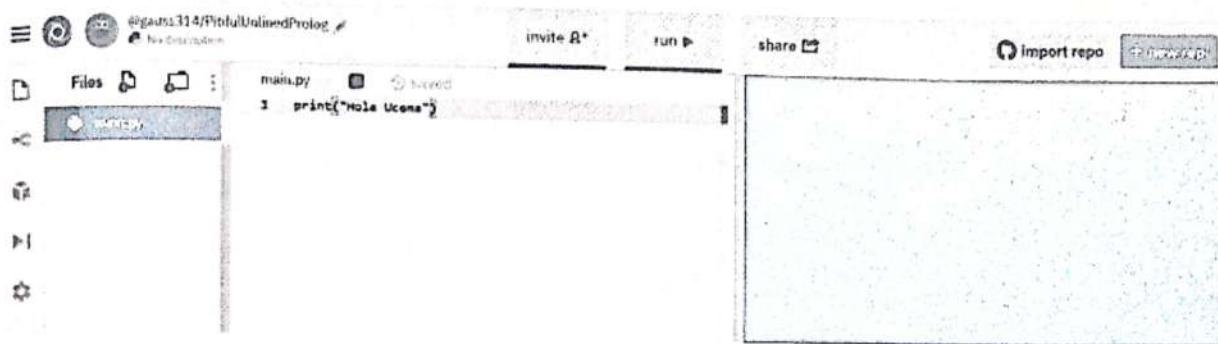


## IDE con intérprete online

Ahora vamos a un intérprete online

Usemos [www.repl.it](http://www.repl.it) Vamos a la web, ponemos nuevo replit, elegimos python y se abre el intérprete online Es un repo gratuito, si se registran pueden guardar sus códigos, es muy útil para usarlo en computadoras donde no tienen anaconda ni python instalado

También es muy útil para cuando quieran probar librerías desconocidas y pesadas, para no andar instalando nada en su compu prueban en estos servicios online y se sacan la duda si vale la pena para instalarlo o no



## Spyder

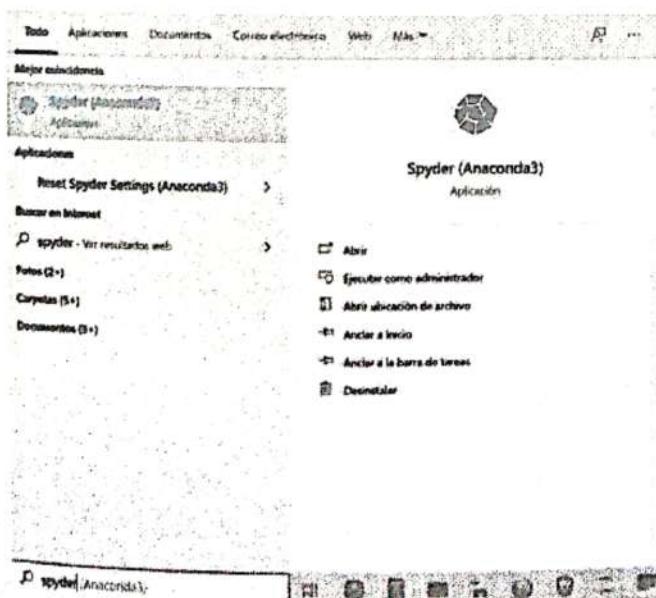
Es el IDE que viene instalado en anaconda, es muy práctico y recomendable para principiantes

¿Cuándo conviene usar spyder o un IDE para programar?

La verdad yo los uso para pequeños scripts separados, es decir armar pequeñas cosas en un solo archivo, ahí es práctico y yo diría lo más recomendable

Para abrirlo buscamos el programa en la barra de windows, una vez abierto ejecutamos el comando de impresión en pantalla

Como verán en el IDE tenemos dos partes, una parte a la izquierda y una a la derecha



Si ejecutan el código en la parte derecha al darle enter se ejecuta, es lo mismo que ejecutarlo en una consola

Pero si lo escriben a la izquierda al darle enter es como si estuviesen escribiendo un texto en word, no se va a ejecutar solo, para que se ejecute le tienen que apretar F5 (o darle al botón de play de la barra de botones)

```

Spyder (Python 3.7)
Archivo Editar Buscar Código fuente Seleccionar Depurar Terminales Proyectos Herramientas Ver Ayuda
C:\Users\floda\spyder-py3
Spyder (Python 3.7)
In [1]: print("Hola Ucema")
Hola Ucema
In [2]:

```

IPython 7.19.2 -- An enhanced Interactive Python.

In [3]: print("Hola Ucema")  
Hola Ucema

00:01:00

conda: base (Python 3.7.5) Line 1, Col 1 UTF-8 CRLF RW Mem 89%

El Spyder es un IDE bastante completo, lo que nos permite no solo editar el código completo línea por línea cómodamente como si escribiesen un archivo de word, sino que tiene resaltado de texto en colores, un buen helper en tiempo real de errores de sintaxis, una consola a la derecha para ir viendo el resultado del código escrito, y muchas cosas más que serán útiles más adelante como inspección de variables y esas cosas

```

Spyder (Python 3.7)
Archivo Editar Buscar Código fuente Seleccionar Depurar Terminales Proyectos Herramientas Ver Ayuda
C:\Users\floda\spyder-py3
Spyder (Python 3.7)
In [1]: print("Hola UCEMA desde Editor")
1 Hola UCEMA desde Editor
2

```

IPython 7.19.2 -- An enhanced Interactive Python.

In [1]: print("Hola Ucema")  
Hola Ucema

In [2]: runfile('C:/Users/floda/.spyder-py3/temp.py', wdir='C:/Users/floda/.spyder-py3')
Hola UCEMA desde Editor

In [3]:

00:02:41

Por ejemplo, si escribimos mal el comando "print" veremos que a la izquierda en donde está el número de línea ya me muestra con una cruz roja que algo mal escribimos antes de ejecutar el código

#### ● Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda  
C:\Users\floida.spyder-py3\temp.py  
temp.py\* □ Terminal 1/A 00:09:01

```
1 print("Hola UCEMA desde Editor")
2
3 print("Hola UCEMA con error")
4
```

Python 3.7.5 (default, Oct 31 2019, 15:18:51) [MSC v.1916 64 bit (AMD64)
Type "copyright", "credits" or "license" for more information.  
IPython 7.10.2 -- An enhanced Interactive Python.  
In [1]: print("Hola Ucema")  
Hola Ucema  
In [2]: runfile('C:/Users/floida/.spyder-py3/temp.py', wdir='C:/Users/floida/.spyder-py3')  
Hola UCEMA desde Editor  
In [3]:

Por último, veamos los helpers, estos son ayudas que nos da el IDE de cada función, si se posicionan con el mouse encima del comando (en este caso encima de print) nos va a desplegar una ayuda super valiosa de todos los argumentos y sus tipos que admite la función

#### ● Spyder (Python 3.7)

Archivo Editar Buscar Código fuente Ejecutar Depurar Terminales Proyectos Herramientas Ver Ayuda  
C:\Users\floida.spyder-py3\temp.py  
temp.py\* □ Terminal 1/A 00:12:71

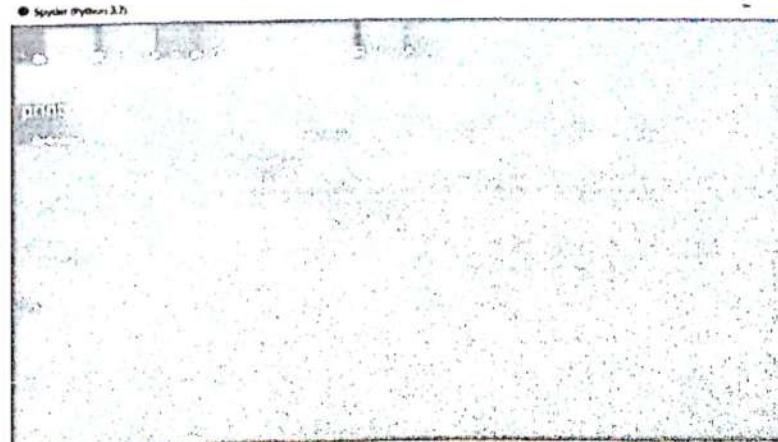
**Definition:** print(\*values: object, sep: Text=..., end: Text=..., file: Optional[\_Writer]=..., flush: bool=...) → None  
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)  
Prints the values to a stream, or to sys.stdout by default. Optional keyword arguments: file: a file-like object (stream); defaults to the current sys.stdout. sep: string between values, default a space. end: string appended after the last value, default a newline. flush: whether to forcibly flush the stream.

```
1 print("Hola UCEMA desde Editor")
2
3 print("Hola.UCEMA.con.error")
4 print(*values: object, sep: Text=..., end: Text=..., file: Optional[_Writer]=..., flush: bool=...) → None
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False) Prints the values
        to a stream, or to sys.stdout by default. Optional keyword arguments: file: a
        file-like object (stream); defaults to the current sys.stdout. sep: string
        inserted between values, default a space. end: string appended after the last
        value, default a newline. flush: whether to forcibly flush the stream.
```

No documentation available  
Click anywhere in this tooltip for additional help

Por último, otra cosa que tiene spyder es que se puede configurar temas oscuros para no dañar la vista cuando estamos muchas horas frente a la pantalla

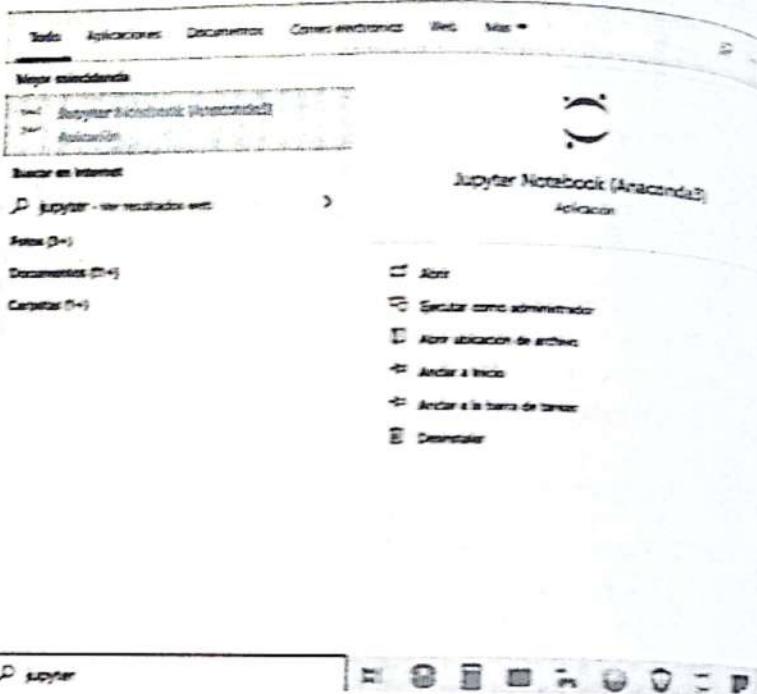
Para ello van a al menú herramientas  
=> Preferencias  
=> Apariencia  
=> Tema de interfaz: Obviamente como verán en "preferencias" se puede configurar de todo



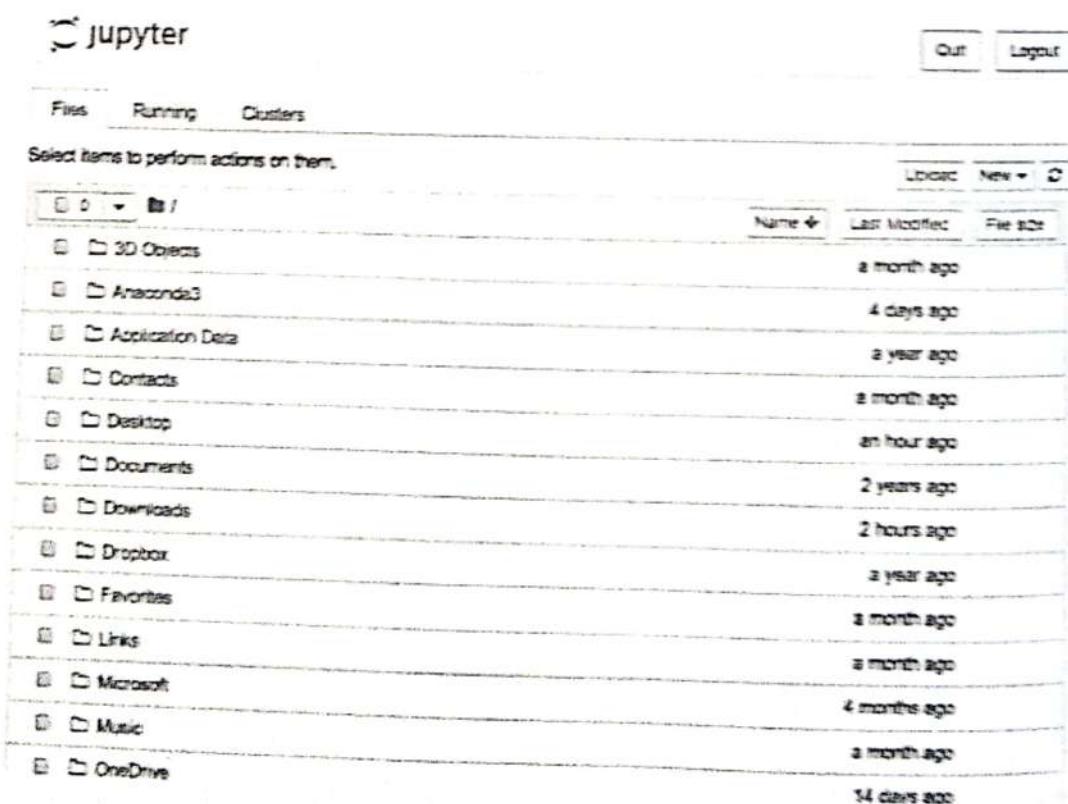
## Jupyter Notebooks

Los notebooks (cuadernos), son herramientas excelentes para aprender ya que permiten mezclar código con anotaciones y tienen una ventaja muy interesante respecto al resto de opciones que es que nos permiten tener mucho código en el mismo archivo y eso, pero con la particularidad de permitirnos ejecutar de a celdas separadas

Para abrir un Jupyter Notebook, lo buscamos en la barra de windows y vemos que es como un navegador de carpetas y archivos



Lo navegamos como si fuera el explorador de archivos de windows, y vamos a la carpeta que queramos o creamos una nueva y de ahí gestionamos nuestros archivos y proyectos, para ello vamos al botón New y elegimos python3 para crear un nuevo notebook de python.

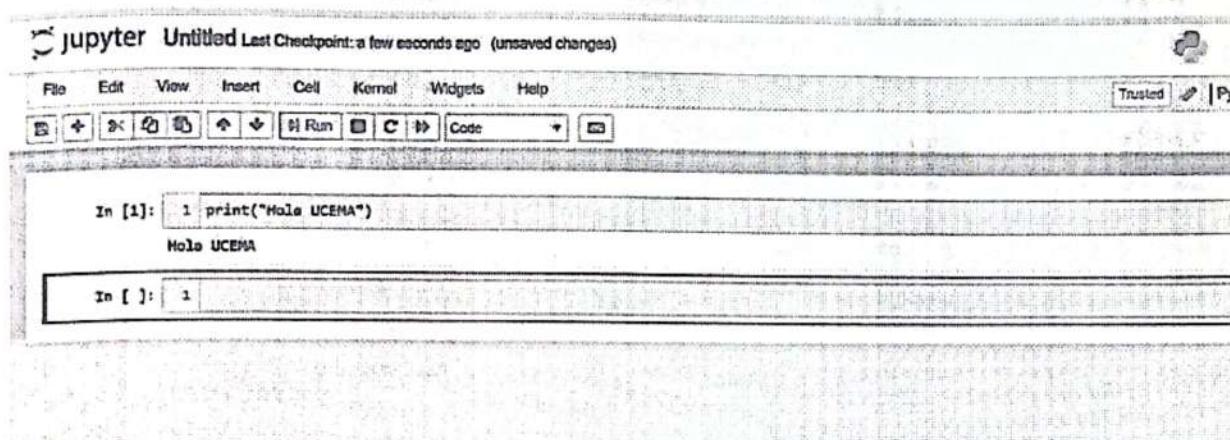


Nota aparte del curso, si quisieran pueden instalar R en anaconda y crear también cuadernos de R

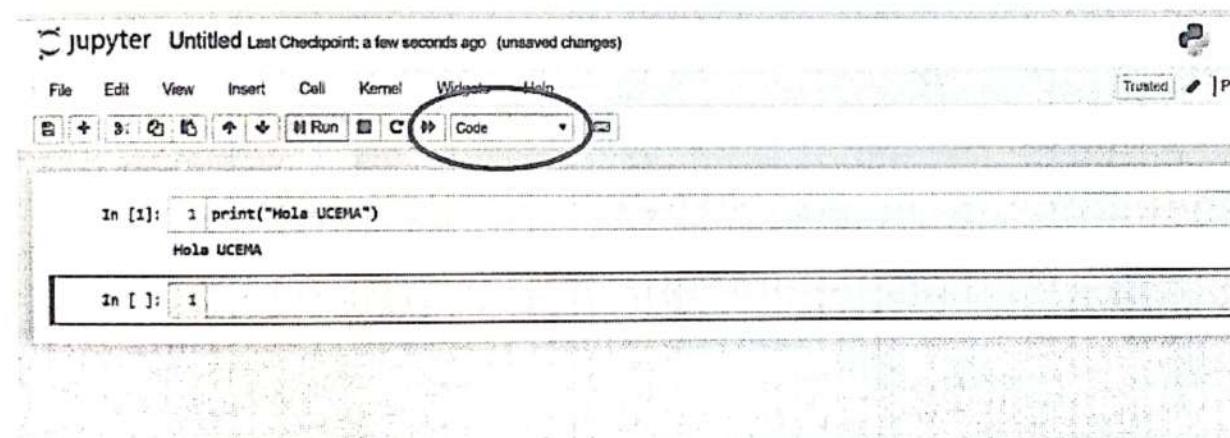
Una vez creado un nuevo Notebook, vemos que tenemos celdas

Las celdas permiten la cantidad de líneas de código que quieran y al ejecutarla se ejecuta todo lo que esté dentro de la celda, para ejecutarlo podemos apretar el botón Run o bien las teclas:

Shift + Enter



Si quieren escribir texto o anotaciones, simplemente seleccionan Markdown del desplegable resaltado en lugar de Code



Respecto al markdown es un estándar para texto enriquecido muy útil y sencillo de usar, googleen si quieren profundizar, acá les dejo lo que más sirve

Si saben html también acepta HTML los notebooks de jupyter

Esto es como se tipea en la celda:

```

1 Esto es un texto así que no se va a ejecutar nada solo me sirve como ayuda
2
3 Puedo usar el estandar markdown para texto enriquecido
4
5 entre guiones bajos es cursiva
6
7 **entre doble asterisco es negrita**
8
9 **Puedo combinar ambas cosas**
10
11 Citas
12 > Esto es una cita
13
14 Listados
15 * Item 1
16 * Item 2
17
18 Links:
19 [Texto del link](http://google.com)
20
21 Encabezados o Títulos
22
23 # H1 - Título principal
24 ## H2 Segundo orden
25 ### H3 Tercer Orden
26 ....
27 ##### H6 Sexto Orden
28

```

Así es como queda impreso el texto enriquecido con Shift + Enter

Esto es un texto así que no se va a ejecutar nada solo me sirve como ayuda

Puedo usar el estandar markdown para texto enriquecido

*entre guiones bajos es cursiva*

**entre doble asterisco es negrita**

**Puedo combinar ambas cosas**

Citas

Esto es una cita

Listados

- Item 1
- Item 2

Links: Texto del link

Encabezados o Títulos

## H1 - Título principal

### H2 Segundo orden

#### H3 Tercer Orden

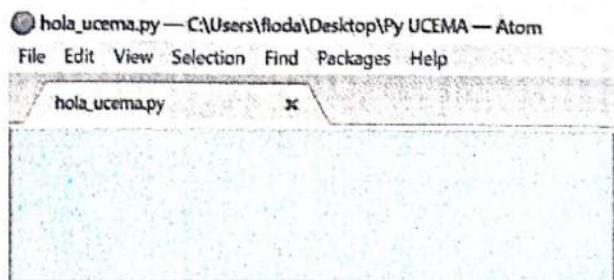
....

**H6 Sexto Orden**

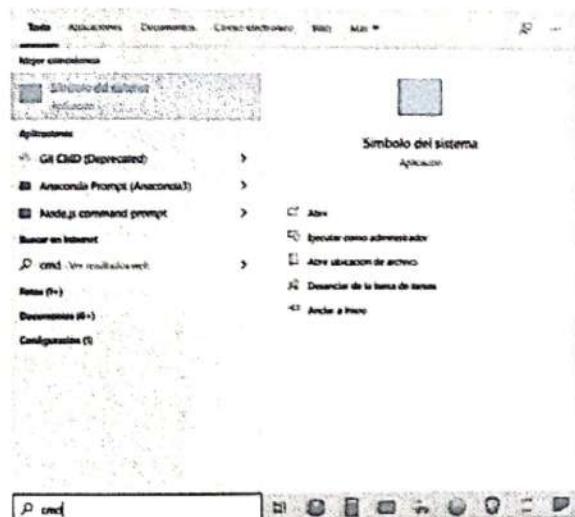
## Editores de código, Atom, SublimeText

Para probar un script super sencillo este método es muy engorroso, pero la realidad es que para proyectos grandes es lo que se usa, no hay nada más cómodo que un buen editor de código, pero bueno, por ahora no vamos a usar este método, pero este es el lugar del curso donde tenía que mostrarlo

Primero abro el editor de código Guardo un archivo `hola_ucema.py`  
En el `hola_ucema.py` escribo el código



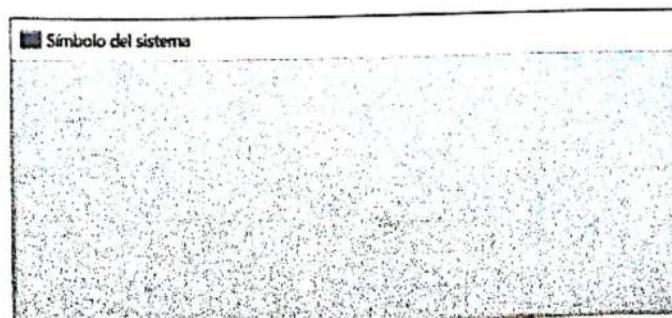
Luego abro la consola buscando CMD



Navego por la consola hasta el directorio del archivo el comando cd pasa al siguiente directorio el comando cd.. vuelve atrás

Una vez en el directorio, lo ejecuto

Para ejecutarlo pongo: `python "nombre_del_archivo"`



## Ejercicios

Escribir el código que devuelva un mensaje cualquiera en pantalla desde los siguientes entornos

- La consola de python
- La consola del IDE spyder
- El editor de código de spyder y correrlo para ver el mensaje en la consola Una linea de Jupyter Notebook
- La consola del intérprete online de repl.it
- El editor del intérprete online de repl.it y correrlo para ver el mensaje en la consola
- Guardar el archivo "mi\_primer\_script.py" desde un editor como atom o sublimeText y ejecutarlo desde la consola de windows

# VARIABLES

Instrucciones: Una instrucción es pedirle al intérprete de python o el lenguaje que usemos que ejecute una función es decir que me dé una "salida" esa salida puede ser una "acción" o simplemente un valor La primera instrucción que vamos a usar es "print()" que lo que hace es imprimir en pantalla el argumento que le pasemos entre paréntesis

```
mi_variable = "Hola Ucema"
print(mi_variable)
```

Hola Ucema

```
mi_variable = 100
print(mi_variable)
```

100

Como ven Para que imprima un texto tengo que ponerlo entre comillas, no así con el número Siempre que nos vayamos a referir a una cadena de texto vamos a ponerla entre comillas

## Asignación de valores a variables

Una variable es como una caja donde uno guarda algo, una vez que pongo algo en la caja, la próxima vez que quiera usar eso que puse, solo tengo que ir a la caja y lo que puse allí dentro, allí estará

```
come = 3
print(come)
```

3

Lo diferente q tienen las variables a las cajas físicas es que puedo poner otra cosa diferente sin tener que sacar lo que tenía previamente, esto es sobreescribir la variable, obviamente que el valor antiguo se pierde

```
ggal = 100
print(ggal)
ggal = 101
print(ggal)
```

100  
101

## Asignación de variables por consola o interfaz de usuario

En algún momento de un programa puede ser necesario tener que pedirle al usuario un dato para guardarlo en una variable, para ello hay muchos método el mas sencillo en python es usar la instrucción input()

```
stopLoss = input("Ingresar el stopLoss ")
print("Tu stopLoss quedó configurado en",stopLoss)
```

Ingresar el stopLoss 100  
Tu stopLoss quedó configurado en 100

Como vemos en este ejemplo ahora a la instrucción print() le pasamos 2 argumentos en vez de 1, un texto "Tu stopLoss quedó configurado en" y el valor que guardamos en la variable

*Siempre que las funciones o instrucciones acepten más de un argumento, estos irán separados por comas*

Otra manera de hacer exactamente lo mismo es concatenando la salida de print, veámoslo en el ejemplo:

```
stopLoss = input("Ingresar el stopLoss ")
print("Tu stopLoss quedó configurado en " + stopLoss)
```

Ingresar el stopLoss 100  
Tu stopLoss quedó configurado en 100

Y otra manera en python de concatenar un texto al valor de una variable es usando "fstrings" son cadenas formateadas y para indicarle al intérprete que queremos referirnos a este tipo de cadenas le poneos la letra f antes de iniciar la cadena. Esto no es muy usado pero viene bien saberlo por si se topan con esto en un código que vean por ahí y lo quieren entender

```
stopLoss = input("Ingresar el stopLoss ")
print(F"Tu stopLoss quedó configurado en {stopLoss}")
```

Ingresar el stopLoss 100  
Tu stopLoss quedó configurado en 100

## Tipos de datos

### Números Enteros int()

```
a = 5  
type(a)
```

int

### Números decimales float()

```
a = 3.14  
type(a)
```

float

### Booleanos (True or False)

```
a = False  
type(a)
```

bool

ojo con la primera letra de True o False, ya que si no es mayúscula Python no lo va a entender, veamos

```
a = True  
type(a)
```

bool

Lo que está intentando hacer es asignarle a la variable "a" el valor contenido en la variable "true" porque entendió que al escribir nosotros true nos referíamos a alguna variable llamada así

```
a = "True"  
type(a)
```

str

obviamente que si ponemos True entre comillas python no va a entenderlo como un valor booleanos sino como un texto cualquiera

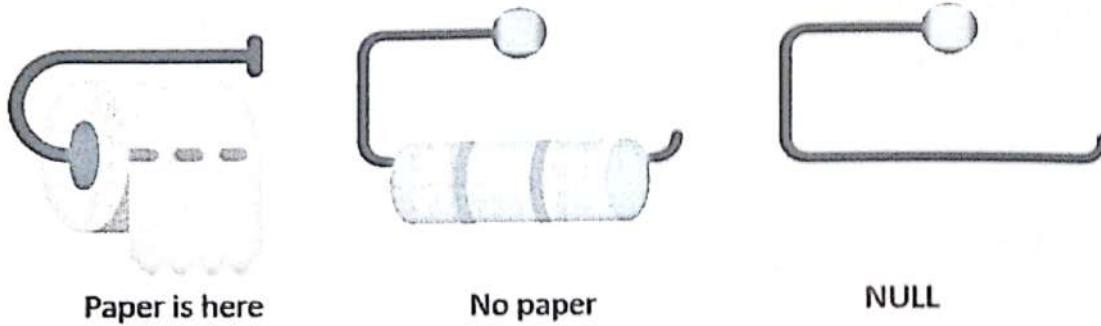
## Nulos

```
a = None
type(a)
```

NoneType

Y eso que fue?

Eso es un dato Nulo, en otros lenguajes se los llama Null que sería como una especie de "no seleccionado" que no es lo mismo que vacío ya que vacío es más un sinónimo de "cero", en cambio el "Nulo" es algo que ni siquiera sabemos si estamos hablando de una variable que representa una fecha o un número o un string por decir algo.. Mas adelante se va a entender mejor el concepto



Obvio que hay muchos tipos mas de variables pero los vamos a ir viendo de a poco a medida que vayamos avanzando, les voy adelantando las mas usadas que veremos en breve:

- Fecha
- Listas
- Tuplas
- Diccionarios
- Objetos

## Ojota con esto, operar variables de diferente tipo

```
a = 2
b = a * 3
print(b)
```

6

```
a = "2"
b = a * 3
print(b)
```

222

## Errores por operar tipos de datos diferentes

```
a = "2"  
b = a + 3  
print(b)
```

```
-----  
TypeError Traceback (most recent call last) <ipython-  
input-16-78260d30c223> in <module> 1 a = "2"  
----> 2 b = a + 3  
TypeError: can only concatenate str (not "int") to str
```

Como vemos, le asigné a una variable un número y a otra un texto, ahora que pasa si las quiero concatenar en una sola oración

Lo que me dice el intérprete de python es que **no puede concatenar strings con enteros**  
Si ambos hubieran sido números el intérprete los hubiera sumado  
Si ambos hubiesen sido texto, los hubiera "concatenado" un texto atrás del otro Pero como le mandamos un número y un texto, no sabe qué hacer y tira error

Si hay algo copado que tiene python es que su intérprete es muy claro al mostrar donde se producen los fallos, aprovechen eso, lean bien el error que tira para entender dónde está fallando nuestra lógica

# Python es un lenguaje interpretado, dinámicamente tipado

## ¿Lo Qué?

Sin entrar en tecnicismos, hay lenguajes de programación compilados y otros interpretados, esto quiere decir que el código que nosotros escribimos para ser ejecutado por la computadora debe primero ser interpretado o compilado

- Los compiladores primero "leen" todo el código, luego lo compilan y recién ahí puede ejecutarse • Los intérpretes en cambio van interpretando el código para que pueda ser ejecutado línea por línea

Los lenguajes interpretados como Python, tienen un intérprete que va "traduciendo" a lenguaje de máquina nuestro código línea por línea mientras se va ejecutando.. Esto quiere decir que en una línea del código yo puedo definir una variable como un "string" o cadena de texto, y un par de líneas más adelante usar esa variable y asignarle un valor numérico "integre" por ejemplo

Esa flexibilidad de poder asignar a una variable, una palabra y en otra línea asignarle un número se la conoce como **Tipado Dinámico**

```
a = "GGAL"
print(type(a))
a = 100
print(type(a))

<class 'str'>
<class 'int'>
```

Si python hubiese sido un lenguaje de tipado estático, yo no hubiera podido asignar a la misma variable primero un tipo de datos y después otro

Por el contrario, un lenguaje compilado, en lugar de tener un "intérprete" que pase a lenguaje de máquina nuestro código línea por línea mientras se va ejecutando, tiene un "compilador" que antes de ejecutar nada hace toda la traducción y luego, una vez traducido (compilado) todo, recién ahí se puede ejecutar

El tipado estático contrariamente al dinámico tiene una ventaja principal que es que al compilarse el código me estaría diciendo el compilador que hay una "incoherencia" en una misma variable a la que le quiero asignar datos que nada que ver y me permitiría corregir esto antes de correr el programa

En Cambio, en un lenguaje interpretado como Python, el intérprete arranca "a ciegas" ejecutando sin tener idea de qué tipo de valor le asignaremos luego una variable y esto es cómodo al principio, pero si nos mandamos cualquiera el código se va a romper en medio de la ejecución

## Operaciones Básicas

```
a = 6
b = 3
suma = a+b
producto = a*b
division = a/b
mod = a%b
potencia = a**b
```

producto

18

mod

0

potencia

216

## Métodos básicos para trabajar con strings

Los métodos mas usuales son

- `string.capitalize()` => Pasa a mayúscula la primera letra de la variable string `string.upper()` => Pasa todas las letras a mayúsculas de la variable string `string.lower()` => Pasa todas las letras a minúsculas de la variable string
- `string.title()` => Pasa las primeras letras de cada palabra a mayuscula de la variable string
- `string.rfind("string_buscado")` => Devuelve la última posición donde aparece por primera vez el `string_buscado` en la variable string
- `string.find("string_buscado")` => Devuelve la primera posición donde aparece por primera vez el `string_buscado` en la variable string
- `string.replace("str_buscado","str_reemplazo")` => Reemplaza "str\_buscado" por "str\_reemplazo" en la variable string
- `string.zfill(largo)` => Rellena con ceros a la izquierda hasta completar el largo pasado como argumento
- `string.count("string_buscado")` => Devuelve la cantidad de veces que aparece el `string_buscado` en la variable string
- `len(string)` => Cuenta la cantidad de caracteres de la variable string
- `string.isalnum()` => True si todos los caracteres son alphanumericos `string.isalpha()` => True si todos los caracteres son letras `string.isdigit()` => True si todos los caracteres son dígitos

## Ejemplos

```
string = "aapl"  
string.upper()
```

'AAPL'

```
string = "El dolar va a valer $211,11"  
string.title()
```

'El Dolar Va A Valer \$211,11'

```
string = "GFGC100.AB"  
string.find(".")
```

7

```
string = "AAPL,GOOGL,AMZN,FB,TSLA"  
string.replace("TSLA","BTC")
```

'AAPL,GOOGL,AMZN,FB,BTC'

```
string="1"  
string.zfill(3)
```

'001'

```
string = "AAPL,GOOGL,AMZN,FB,TSLA"  
string.count("A")
```

4

```
string = "BTC"  
len(string)
```

3

```
string = "123"  
string.isdigit()
```

True

```
string = "123"  
string.isalnum()
```

True

```
string = "123"  
string.isalpha()
```

False

## Cambiando los tipos de datos

Generalmente este tipo de cosas de cambiar el tipo de datos que tiene una variable no es una gran idea en programación ya que seguramente si nos viene una variable en un tipo de dato (entero, flotante, texto etc) es porque en algún momento el programa estará esperando ese tipo y si en el medio se lo cambiamos probablemente más adelante aparezcan problemas, esto claro está cuando trabajamos en proyectos grandes, si son pequeños programas caseros donde todo el código lo escribimos nosotros mismos, no hay problema, y suele ser un recurso muy usado...

Dicho esto, ya puedo dormir tranquilo que les dije que no era buena idea así que veamos esta joyita que permite python

```
precio = 100.23433
print(precio,type(precio))

100.23433 <class 'float'>
```

```
precio = 100.23433
precio = int(precio)
print(precio,type(precio))

100 <class 'int'>
```

```
precio = 100.23433
precio = str(precio)
print(precio,type(precio))

100.23433 <class 'str'>
```

Pero ojo, siempre recuerden a Peter Parker: "Todo gran poder conlleva una gran responsabilidad"

Esto de andar jugando demasiado con los tipos de datos puede generar cosas inesperadas y de consecuencias impredecibles

```
precio * 2
'100.23433100.23433'
```

## Palabras Reservadas

Ojo al piojo con las palabras reservadas, son pocas pero son palabras que python se reserva para "uso especial"

Esto quiere decir que no podemos usar esas palabras para llamar a variables o funciones etc

Listado de palabras reservadas
and as assert break class continue def del elif else
except exec finally for from global if import in is
lambda not or pass print raise return try while with

veamos qué pasa si le quiero poner de nombre a una variable una palabra reservada

```
global = 12  
print(global)
```

```
File "<ipython-input-38-1f5e9f642b7a>", line 1  
SyntaxError: invalid syntax
```

## Números al Azar

Muchas veces en nuestros programas vamos a necesitar tomar determinados valores arbitrarios, para ello contamos con las siguientes funciones

### Valor flotante al azar entre 0 y 1

```
import random  
random.random()  
0.5291112005019688
```

### Valor flotante entre min y max

```
random.uniform(2,3)  
2.432532146642264
```

### Valor entre min y max definiendo intervalo o "step"

```
min = 0  
max = 100  
paso = 2  
random.randrange(min,max,paso)
```

# Número aleatorio en una Distribución

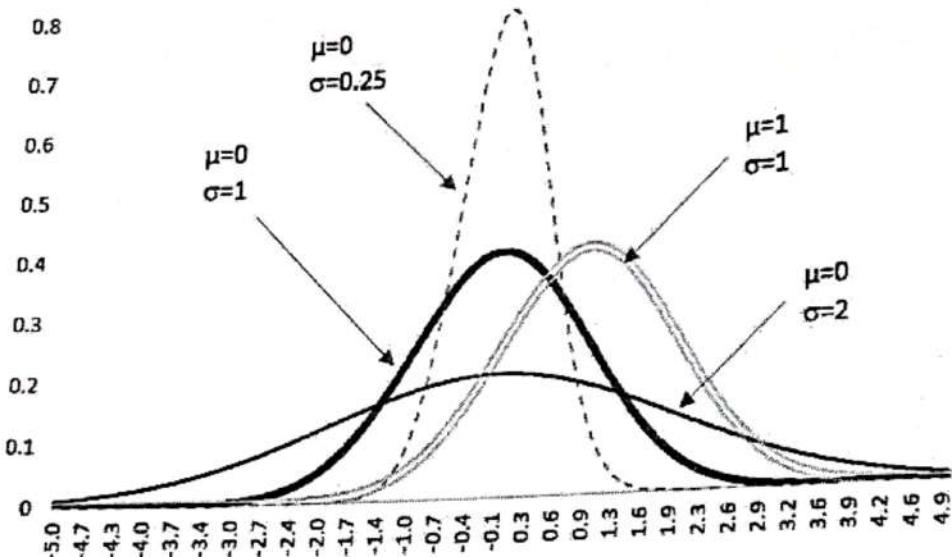
A parte de darnos números aleatorios al azar, la librería random nos provee de funciones para obtener un valor aleatorio de una función de distribución de probabilidades como por ejemplo de la famosa distribución normal, esto nos va a ser sumamente útil para realizar más adelante simulaciones de montecarlo

Repasemos primer que es una distribución normal

Lo que muestra esta función en el eje "Y" es la probabilidad de ocurrencia de un valor del eje "X", a mas alto el punto en la gráfica más probable es.

Podremos ver los valores del eje "X" como +/- la cantidad de desvíos estándar ( $\sigma$ ) de la media ( $\mu$ )

Por ejemplo, si asumimos que los retornos diarios de una acción siguen una distribución normal,  $\mu$  sería el retorno medio y  $\sigma$  la volatilidad de los retornos



Utilizando la función `random.normalvariate(mu,sigma)` vamos a obtener valores aleatorios del eje X de la nominal ( $\mu$ ,  $\sigma$ )

```
mu = 0
sigma = 1
random.normalvariate(mu,sigma)
```

-0.06611273222951637

Otras distribuciones posibles son

- `random.betavariate(alfa,beta)`
- `random.gamavariate(alfa,beta)`
- `random.expovariate(lambda)`
- `random.lognormalvariate(mu,sigma)`
- `random.paretovariate(alfa)`
- `random.weibullvariate(alfa,beta)`

# Operaciones Matemáticas. Librería Math

```
import math
```

## Constantes

```
math.pi  
3.141592653589793
```

```
e = math.e  
e  
2.718281828459045
```

## Redondeo y truncado

Entero inmediato inferior

```
math.floor(5.44)
```

5

Entero inmediato superior

```
math.ceil(5.44)
```

6

Entero

```
math.trunc(-5.34)
```

-5

Redondear por cantidad de decimales

```
round(5.76543, 3)
```

5.765

## Logaritmos

Logaritmo base 10 (Si se pasa un segundo argumento a la función es la base, por default es e)

```
math.log(100,10)
```

2.0

```
e = math.exp(1)
math.log(e)
```

1.0

## Otras Funciones

Máximo común Divisor, factorial, etc.

```
math.gcd(9,24)
```

3

```
math.factorial(6)
```

720

## isClose

Decidir si un numero está cerca de otro o no en función de una tolerancia porcentual

```
a = 989
b = 1000
tol= 0.05
math.isclose(a,b,rel_tol=0.01)
```

False

## Ejercicios

1 - Obtener un precio aleatorio para la acción de GGAL que oscile entre 100 y 110 y con precisión de 0.01, asignarlo a la variable ggal

Como habrán notado el principal problema es que la función randrange permite sensibilidades enteras no de 1/100 como pide el ejercicio, así que tenemos que usar el ingenio para adaptar la herramienta a nuestro problema

3 - Asumiendo que un activo tiene una distribución de sus rendimientos diarios muy similar a la distribución normal, y que su media diaria es de 0.1% con una volatilidad diaria del 2.5%. Escribir la instrucción que me devuelva valores aleatorios de su rendimiento diario

## Respuestas

Un primer camino (el que uso yo siempre) es jugar con las unidades, imaginar el precio en lugar de en centavos, y luego si volver a pasarlo una vez resuelto el problema y listo, quedaría así la solución

```
#-----#
# Rta Ejercicio 1 #
# ----- #

import random
min = 10000
max = 11000
step = 1
ggal = random.randrange(min,max,step)/100
ggal
```

102.62

Otra alternativa sería separando el problema en dos, buscar un aleatorio entero entre 100 y 109 y otro aleatorio decimal entre 0 y 1 y sumarlos luego. Esto conlleva mayor esfuerzo computacional por lo que en este caso no es la mejor alternativa ya que la solución mostrada anteriormente es más simple y requiere menos esfuerzo computacional.

Esto del esfuerzo computacional en un ejemplito simple, así como este no tiene sentido plantearlo, pero imaginen que "ese precio aleatorio" es algo que voy a usar en una simulación que repite el proceso miles de veces para cada fecha y en varias fechas para cada activo y lo hace diariamente con miles de activos

```
# Otra alternativa
#-----#
# Rta Ejercicio 1 #
#----- #

min = 100
max = 109
step = 1
entero = random.randrange(min,max,1)
decimales = random.uniform(0,1)
ggal = round(entero + decimales,2)
ggal
```

107.5

```
#-----#
# Rta Ejercicio 3 #
#----- #

import random
random.normalvariate(0,2.5)

-4.380819654141598
```

# Trabajando con Fechas

## La librería datetime

Primero debo importar la librería datetime para trabajar fechas y horas, es una librería de las más estándar de python

```
import datetime as dt
```

Y que significa el "as dt" bueno esto es un "alias" es decir que cuando nos queramos referir a la librería datetime la llamaremos dt

Como vemos hasta ahora el importar una librería no me devuelve nada, solo carga en el kernel las funciones de la librería

Antes de avanzar con los métodos de esta librería vamos a ver un concepto importante de las librerías que son las sus librerías, o paquetes individuales de la librería maestra En el caso de la librería **datetime** tenemos los siguientes paquetes o su librería:

- date
- time
- datetime
- timedelta
- timezone
- tzinfo

## Importando sub-paquetes

Importemos una sublibreria de la librería principal **datetime**, por ejemplo, de la siguiente manera:

```
from datetime import datetime as dt_dt
from datetime import date as dt_date
```

Esos alias nos permiten referirnos en modo más abreviado a los sub-paquetes:

• En lugar de **datetime.datetime** solo diremos **dt\_dt** •

En lugar de **datetime.date** solo diremos **dt\_date**

Y en lugar de importar todo el paquete datetime solo importamos esos dos sub-paquetes Bueno, vamos a ver cuál es la fecha actual con la hora actual

```
hoy = dt_dt.now()
print(hoy,type(hoy))
```

```
2020-03-13 16:34:55.435794 <class 'datetime.datetime'>
```

como vemos, el dato de la fecha almacenado en la variable "hoy" es un dato de tipo objeto, ya veremos más adelante en detalle el significado de un "objeto" en programación, pero por ahora vamos a decir que a diferencia de una variable un objeto además de un valor único tiene atributos y métodos

si en lugar de now() usamos el método today() de la misma librería obtenemos el mismo resultado

```
hoy = dt_dt.today()
print(hoy,type(hoy))
```

2020-03-13 16:46:59.125873 <class 'datetime.datetime'>

*¿Y si quiero solo la fecha y no me importa la hora?*

Para ello usaremos una sublibreria de datetime que es date y su método today() que se llama igual que en la sublibreria datetime

```
hoy = dt_date.today()
print(hoy,type(hoy))
```

2020-03-13 <class 'datetime.date'>

¿Pero qué tipo de dato es en realidad ese? Bueno, ya llegaremos, pero acá estamos hablando de objetos. Una de las ventajas de tener las fechas como objetos es que podemos aplicarles todos los métodos de librerías como datetime, uno de ellos puede ser calcular diferencias entre fechas, por ejemplo

### Fechas en formato string

In [41]:

```
hoy = dt.datetime.today().ctime()
print(hoy,type(hoy))
```

Fri Mar 13 16:40:57 2020 <class 'str'>

# Convertir un string en un Objeto de DateTime

¿y para que querría hacer tal cosa? si alguien se hizo esta pregunta, excelente  
Siempre hay que hacerse esa pregunta antes de volverse loco buscando el método o  
la función que lo permita

En este caso recontra vale la pena la ecuación costo-beneficio de hacer esta conversión

El tema es que las fechas en modo "string" no me sirven de mucho más que para mostrar en pantalla, si quiero saber la diferencia de días, minutos o lo que fuera entre dos fechas, las necesito como objetos de fecha

```
hoy = dt_dt.today()
expiracion_str = "2020-04-17"
expiracion = dt_dt.strptime(expiracion_str, '%Y-%m-%d')
expiracion
datetime.datetime(2020, 4, 17, 0, 0)
```

Ahora que ya tengo la fecha de hoy y la de expiración en el mismo tipo de objeto, puedo calcular la diferencia entre ambas como una simple resta

```
expiracion - hoy
```

```
datetime.timedelta(days=34, seconds=26251, microseconds=602673)
```

Como vemos el resultado de esa resta es un objeto de la librería `timedelta` que como dijimos es un sub-paquetes de `datetime`

Ahora si queremos obtener los días o los segundos de ese objeto en una variables simplemente acudimos al atributo `days` o al atributo `seconds`

```
dias = (expiracion - hoy).days
segundos = (expiracion - hoy).seconds
print("Días restantes:", dias, "\nSegundos restantes:", segundos)
```

Días restantes: 36

Segundos restantes: 63156

Y si queremos acceder al string de la expiración que ya convertimos en objeto, podemos usar el método `ctime()`

```
'Fri Apr 17 00:00:00 2020'
```

```
dt_dt.ctime(expiracion)
```

## Generar una fecha como objeto de datetime

Otra forma de generar una fecha como objeto de datetime en lugar de partir de un string puede ser partiendo de los valores de año, mes y día

```
año = 2020
mes = 12
dia = 24
hora = 23
minutos = 59
segundos = 59
fecha = dt_dt(año,mes,dia,hora,minutos, segundos)
fecha, fecha.ctime()
(datetime.datetime(2020, 12, 24, 23, 59, 59), 'Thu Dec 24 23:59:59 2020')
```

Luego el paso inverso siempre es más fácil

**Convertir a texto algo es simplemente usar la función str()**

```
str(expiracion)
'2020-04-17 00:00:00'
```

## Distintos formatos para mostrar fechas

Claramente los formatos de fecha usados en bases de datos no siempre son cómodos de visualizar, de hecho, generalmente son muy incómodos, por lo que es común usar en interfaces de usuarios formatos más amigables a la lectura humana, para ello vamos a usar la función strftime para pasar el objeto fecha a un string cómodo de leer

```
from datetime import datetime as dt_dt
hoy = dt_dt.today()
hoy_txt = dt_dt.strftime(hoy, "%d %B, %Y")
hoy_txt
```

'13 March, 2020'

## Seteo de parametrización local

La Librería `locale` nos permite setear a los usos locales los atributos, entre otras de:

- Fechas y horas: `locale.LC_TIME`
- Signos y nomenclaturas monetarias: `locale.LC_MONETARY`
- Numeración y signos de puntuación: `locale.LC_NUMERIC`

```
import locale
from datetime import datetime as dt_dt

locale.setlocale(locale.LC_TIME, "en")
hoy = dt_dt.today()
hoy = dt_dt.strftime(hoy, "%A %d %B, %Y")

locale.setlocale(locale.LC_TIME, "esp")
hoy_es = dt_dt.today()
hoy_es = dt_dt.strftime(hoy_es, "%A %d de %B, %Y")

hoy, hoy_es
```

('Friday 13 March, 2020', 'viernes 13 de marzo, 2020')

Listado de posibles configuraciones para `strftime` y `strptime`

### Seteos para Fechas

#### Fechas

Código	Significado	Ejemplo
%a	Nombre del día abreviado	Mon
%A	Nombre del día de semana, 0=Domingo... 6=sábado	Monday
%d	Numero de día del mes rellenado con 0	08
%-d	Numero de día del mes sin rellenar con 0	8
%b	Nombre del mes abreviado	Jan
%B	Nombre del mes	January
%m	Numero de mes del mes rellenado con 0	08
%-m	Numero de mes del mes sin rellenar con 0	8
%y	Numero año sin siglo	19
%Y	Numero año con siglo	2019
%j	Número de día del año rellenado con ceros	031
%-j	Número de día del año sin ceros	31
%U	Número de semana del año (Domingo=Primer día de semana)	52
%W	Número de semana del año (Lunes=Primer día de semana)	52
%c	Representación completa	Mon Sep 30 07:06:05 2013
%x	mes/día/año	07/24/2020

Listado de posibles configuraciones para `srtftime` y `srtptime`**Seteos para Horarios****Hora, minutos y segundos**

Código	Significado	Ejemplo
%H	Numero de Hora (0-24) con ceros	09
%-H	Numero de Hora (0-24) sin ceros	9
%I	Numero de Hora (0-12) con ceros	09
%-I	Numero de Hora (0-12) sin ceros	9
%p	AM/PM	AM
%M	Minutos con ceros	09
%-M	Minutos sin ceros	9
%S	Segundos con ceros	05
%f	Microsegundos	123456
%X	Hora:minutos:segundos	20:12:55
%H-	numero de hora, + caracteres	20-

Nota, en algunos sistemas operativos en lugar del "-" funciona el "#"

## ¿Qué es un timestamp?

El timestamp más usado es el número de segundos que han transcurrido desde las 0 horas del 1 de enero de 1970 GMT.

O sea que es una cantidad de segundos que nos sirve para identificar una fecha y hora exactas con un simple número entero y su uso es muy común en informática ya que ocupa poco espacio en una base de datos y es universalmente conocido.

También se lo conoce como:

- Fecha "Epoch"
- Fecha UNIX o tiempo UNIX

Como curiosidad si quieren saber la fecha actual en este formato visiten

- <https://www.epochconverter.com/> (<https://www.epochconverter.com/>)
- <https://currentmillis.com/> (<https://currentmillis.com/>)

## Obtener el timestamp de un objeto fecha

```
hoy = dt_dt.today()  
timestamp = hoy.timestamp()  
print(timestamp)
```

1584131430.513358

## Obtener el objeto fecha de un timestamp

```
timestamp = 1584131430.513358  
fecha = dt_dt.fromtimestamp(timestamp)  
fecha
```

datetime.datetime(2020, 3, 13, 17, 30, 513358)

## Pasando a Otro huso horario

```
import pytz
from datetime import datetime
as dt_dt hoy = dt_dt.now()
hoy_berlin = hoy.astimezone(pytz.timezone('Europe/Berlin'))
print("La hora local es", hoy.ctime())
print("La hora de Berlin es", hoy_berlin.ctime())
```

La hora local es Sat Mar 14 18:37:52 2020  
 La hora de Berlin es Sat Mar 14 22:37:52 2020

Dejo comando para listar todas las zonas mas usuales, imprimo solo las primeras 10, pero si quieren ver todas pongan directamente

- `pytz.common_timezones`

```
pytz.common_timezones[0:10]
['África/Abidjan', 'África/Accra', 'África/Addis_Ababa', 'África/Algiers',
'África/Asmara', 'África/Bamako', 'África/Bangui', 'África/Banjul',
'África/Bissau', 'África/Blantyre']
```

¿Y qué otras funciones tienen esta librería `datetime`?

Muchas más, obvio no vamos a mostrar todas, pero acostúmbrense a visitar la documentación de las librerías que usan ya que muchas veces les va a simplificar la vida

En este caso se trata de una librería estándar de python así que vamos a encontrar su documentación en [docs.python.org](https://docs.python.org/3/library/datetime.html) pero si no fuera el caso nuestro mejor aliado siempre es Google

<https://docs.python.org/3/library/datetime.html> (<https://docs.python.org/3/library/datetime.html>)

# Jugando con Fechas y Calendarios

*Podemos imprimir en pantalla un calendario*

`import calendar`

```
año = 2020
separacion_horiz = 1
separacion_vert = 1
calle = 3
meses_por_fila = 3
calendar.prcal(año, separacion_horiz, separacion_vert, calle, meses_por_fila)
```

2020

January

Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5			1	2					1
6	7	8	9	10	11	12	3	4	5	6	7	8	9
13	14	15	16	17	18	19	10	11	12	13	14	15	16
20	21	22	23	24	25	26	17	18	19	20	21	22	23
27	28	29	30	31			24	25	26	27	28	29	
													30
													31

February

March

April

May

June

July

August

September

October

November

December

Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
														1	2	3	4	5	6	
1	2	3	4	5						1	2			1	2	3	4	5	6	
6	7	8	9	10	11	12	3	4	5	6	7	8	9	7	8	9	10	11	12	13
13	14	15	16	17	18	19	10	11	12	13	14	15	16	14	15	16	17	18	19	20
20	21	22	23	24	25	26	17	18	19	20	21	22	23	21	22	23	24	25	26	27
27	28	29	30	31			24	25	26	27	28	29	30	28	29	30				

O podemos imprimir solo un mes cualquiera

```
año = 2020  
mes = 3  
calendar.pmonth(año, mes)
```

```
March 2020  
Mo Tu We Th Fr Sa Su  
      1  
 2  3  4  5  6  7  8  
 9 10 11 12 13 14 15  
16 17 18 19 20 21 22  
23 24 25 26 27 28 29  
30 31
```

#### Años bisiestos

Por ejemplo, vamos a preguntar si un año es bisiesto o no

Dentro de calendar ya está predefinido un método que se llama `isleap()`  
Y ese método admite un argumento (en este caso obligatorio que es el año en cuestión)

```
calendar.isleap(2020)
```

True

También podemos saber la cantidad de días/años bisiestos entre dos años

```
calendar.leapdays(2020, 2035)
```

4

#### Funciones básicas de calendar

Si queremos saber que día de la semana caerá una fecha (Por default está seteado 0=lunes, 1=martes, y así hasta 6=Domingo)

```
dia = calendar.weekday(2020, 3, 27)  
print(dia)
```

4

Sigue funcionando el seteo de la librería locale para las funciones de calendar

```
import locale  
locale.setlocale(locale.LC_TIME, "en")  
calendar.day_name[dia]  
'Friday'
```

```
locale.setlocale(locale.LC_TIME, "esp")  
calendar.day_name[dia]  
'viernes'
```

#### Función monthrange

Esta función me devuelve el número de día (lunes=0... domingo=6) del primer día del mes, y la cantidad de días del mes

```
locale.setlocale(locale.LC_TIME, "esp")  
calendar.monthrange(2020, 3)
```

(6, 31)

Ante cualquier duda de los métodos de esta librería, ver la documentación oficial:

<https://docs.python.org/3/library/calendar.html> (<https://docs.python.org/3/library/calendar.html>)

## Ejercicios

- 1- Hacer un script que devuelva en la variable año, el año actual
- 2- Escribir el código que asigne un día y mes definidos en las primeras líneas en las variables día y mes respectivamente, y que devuelva que día de la semana cae esa fecha en el año actual
- 3- Hacer un script que le informe al usuario cuantos días faltan para terminar el año
- 4- Hacer un script que le informe al usuario que fracción de año queda para la expiración de una opción que vence el 18 de diciembre del corriente año (que funcione sea cual sea el año que se ejecute el código)

## Respuestas

In [19]:

```
#-----#
# Rta Ejercicio 1 #
# ----- #

from datetime import datetime
as dt_dt hoy = dt_dt.now()
año = hoy.year
año
```

2020

```
#-----#
# Rta Ejercicio 2 #
# ----- #

import calendar
from datetime import datetime as dt_dt
hoy = dt_dt.now()
año = hoy.year

mes = 11
dia = 6

diaSemana = calendar.weekday(2020,mes,dia)
dia_nombre = calendar.day_name[diaSemana]
print("Este año cae un: ",dia_nombre)
```

Este año cae un: Friday

```
#-----#
# Rta Ejercicio 3 #
#-----#
from datetime import datetime as dt_dt
hoy = dt_dt.now()
año = hoy.year
fin_año = dt_dt(año,12,31)
dias_faltantes = (fin_año - hoy).days
dias_faltantes
```

288

```
#-----#
# Rta Ejercicio 4 #
#-----#
from datetime import datetime as dt_dt
hoy = dt_dt.now()
año = hoy.year
expiracion = dt_dt(año,12,18)
dias_faltantes = (fin_año - hoy).days
fraccion_año = dias_faltantes/365
fraccion_año
```

0.7890410958904109

# Otras estructuras de datos en python

## Listas

Las listas son simplemente un conjunto de datos y se lo define entre corchetes con sus elementos separados con coma  
 Es lo que, en otros lenguajes como Java, Php, C++ se los conoce como Arrays  
 o Arreglos Una lista vacía se la escribe así

```
lista_vacia = list()
lista_vacia
```

[]

O así

```
lista_vacia = []
lista_vacia
```

[]

Una lista con los números del 1 al 4 se la puede definir así:

```
lista = [1,2,3,4]
print(lista)
```

[1, 2, 3, 4]

O así

```
lista = list([1,2,3,4])
print(lista)
```

[1, 2, 3, 4]

Si quisiera poner dentro de la lista elementos de distinto tipo no hay problema

```
listado = ["CERO",1,2,"tres",4.0,True]
listado
```

['CERO', 1, 2, 'tres', 4.0, True]

¿Y qué pasa si le preguntamos al intérprete de que tipo es la variable listado?

```
listado = ["CERO",1,2,"tres",4.0,True]
type(listado)
```

list

Para acceder a un elemento cualquiera de una lista tengo que referirme al elemento entre corchetes Recordar siempre que el primer elemento de una lista es el número CERO

```
listado = ["CERO",1,2,"tres",4.0,True]
print(listado[0])
```

CERO

Así como el primer elemento es el 0, si sigo para atrás al -1 es como empezar de derecha a izquierda desde el final, como si fuera circular la lista

```
listado = ["CERO",1,2,"tres",4.0,True]
print(listado[-1])
```

True

```
listado = ["CERO",1,2,"tres",4.0,True]
print(listado[-2])
```

4.0

## Slicing de una lista

Para recortar un "pedazo" de la lista accedo con el caracter ":" A la izquierda de los ":" va el "DESDE" A la derecha de los ":" va el "HASTA"

Recordemos siempre que el primer elemento es el 0

```
listado = ["CERO", 1, 2, "tres", 4.0, True]
listado[2:4]
```

[2, 'tres']

Como verán el DESDE es inclusive y el HASTA es no inclusive

### ¿Qué pasa con [x:]?

Va desde el elemento x inclusive hasta el final

```
listado = ["CERO", 1, 2, "tres", 4.0, True]
listado[2:]
[2, 'tres', 4.0, True]
```

### ¿Qué pasa con [:x]?

Va desde el 1º elemento de la lista hasta el número x no inclusive

```
listado = ["CERO", 1, 2, "tres", 4.0, True]
listado[:2]
```

['CERO', 1]

### ¿Qué pasa con [:]?

Son todos los elementos

```
listado = ["CERO", 1, 2, "tres", 4.0, True]
listado[:]
```

['CERO', 1, 2, 'tres', 4.0, True]

## ¿Qué pasa con los números negativos [-x:-y]?

En lugar de contar de izquierda a derecha cuentan de derecha a izquierda, veamos ejemplos

```
listado = ["CERO",1,2,"tres",4.0,True]
listado[:-2]
```

[ 'CERO', 1, 2, 'tres' ]

```
listado = ["CERO",1,2,"tres",4.0,True]
listado[-2:]
```

[4.0, True]

```
listado = ["CERO",1,2,"tres",4.0,True]
listado[-4:-1]
```

[2, 'tres', 4.0]

## Un Tercer argumento para el slicing: El paso [desde:hasta:paso]

El paso define cada cuanto avanza el cursor para tomar el siguiente elemento, por default sino se indica es claramente 1.

```
listado = [0,1,2,3,4,5,6,7,8,9,10,11,12]
listado[3::2]
```

[3, 5, 7, 9, 11]

En ese ejemplo le pido que vaya desde el 3º elemento hasta el final, y por último que vaya de a 2 en 2, es decir que, del 3, salta al 5 y así.

Veamos otros ejemplos

Si usamos por ejemplo [::3] al no poner ni desde ni hasta le decimos que agarre todos, y el 3 del final le estamos diciendo que vaya saltando de a 3 en 3

```
listado = [0,1,2,3,4,5,6,7,8,9,10,11,12]
listado[::3]
```

[0, 3, 6, 9, 12]

Por último, veamos un caso muy usado que es el `[::-1]`, como se imaginarán lo que hace es recorrer todo, pero al revés es decir devuelve la lista "dada vuelta"

```
listado = [0,1,2,3,4,5,6,7,8,9,10,11,12]
listado[::-1]
```

```
[12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

## Longitud de una lista

Para saber la cantidad total de elementos de una lista usamos la función `len`

```
listado = ["CERO",1,2,"tres",4.0,True]
len(listado)
```

6

## El típico mensaje de "Index out of range"

Cuando recorremos los elementos de una lista, lo primero que tenemos que pensar es que existan los elementos que busquemos porque si le pedimos a python que nos devuelva un numero de elemento que la lista no tiene, nos va a tirar un error y se va a cortar el programa en ese momento (ya que python no sabe que hacer)

```
listado = ["CERO",1,2,"tres",4.0,True]
print(type(listado[12]))
```

---

```
IndexError Traceback (most recent call last)
<ipython-input-20-7876f73ae61a> in <module>
  1 listado = ["CERO",1,2,"tres",4.0,True]
----> 2 print(type(listado[12]))
```

```
IndexError: list index out of range
```

## Cambiando algún valor de la lista

En el siguiente ejemplo le vamos a cambiar el segundo elemento (es decir el de índice 1, recuerden que arranca de 0)

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
listado[1]=["COME"]
print(listado)
['GGAL', 'COME', 'YPFD', 'CEPU', 'EDN', 'LOMA', 'CRES']
```

## Cambiando varios valores de la lista por un solo valor

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
listado[1:3]=["COME"]
print(listado)
['GGAL', 'COME', 'CEPU', 'EDN', 'LOMA', 'CRES']
```

¿Y si quiero cambiar los dos elementos por COME?

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
listado[1:3]=["COME", "COME"]
print(listado)
['GGAL', 'COME', 'COME', 'CEPU', 'EDN', 'LOMA', 'CRES']
```

## Eligiendo valores al azar de una lista

Volvemos a usar la librería Random que usamos la clase pasada

```
import random
```

Elegimos 1 solo valor al azar

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
random.choice(listado)
'PAMP'
```

elegimos "n" valores al azar, sin repetir, es decir "n" tiene que ser menor o igual a la cantidad de

elementos de la lista

```
random.sample(listado,3)  
['YPFD', 'GGAL', 'CRES']
```

Reordenamos aleatoriamente la lista

```
random.shuffle(listado)  
print(listado)  
['LOMA', 'GGAL', 'YPFD', 'CRES', 'CEPU', 'PAMP', 'EDN']
```

Ya vamos a ver más funciones de la librería random, pero por ahora sigamos con las tuplas

## Tuplas

Las tuplas son una estructura similar a las listas, pero son inmutables, es decir que una vez que las definimos, ya no les podemos cambiar los valores

A diferencia de las listas las tuplas se escriben entre paréntesis

```
listado = ("GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES")  
type(listado)
```

tuple

```
listado = ("GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES")  
print(listado[1])
```

PAMP

Acostúmbrense a leer los errores, vamos a provocar el error de querer asignar un nuevo valor a un elemento de una tupla

```
listado = ("GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES")  
listado[1]=["COME"]
```

```
TypeError Traceback (most recent call last)  
<ipython-input-23-a13bdbe480fb> in <module>  
 1     listado = ("GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES")  
----> 2 listado[1]=["COME"]
```

```
TypeError: 'tuple' object does not support item assignment
```

¿Pero y si por "x" razón tengo una tupla y le quiero cambiar un valor? Python

tiene un método para pasar de tupla a lista `list()` y viceversa `tuple()`

```
listado = ("GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES")
print(type(listado))
listado = list(listado)
print(type(listado))
listado = tuple(listado)
print(type(listado))

<class 'tuple'>
<class 'list'>
<class 'tuple'>
```

## Métodos de Tuplas y Listas

### Min y Max

```
cantidades = (4, 2, 5, 6, 8, 10, 3, 5, 7, 4, 2, 6)
print(min(cantidades))
```

2

```
cantidades = (4, 2, 5, 6, 8, 10, 3, 5, 7, 4, 2, 6)
print(max(cantidades))
```

10

### Index

Obtenemos el índice dentro de la lista del elemento pasado como argumento (la primera aparición si esta repetido)

```
cantidades = (4, 2, 5, 6, 8, 10, 3, 5, 7, 4, 2, 6)
cantidades.count(4)
```

0

### Count

Cuenta la cantidad de apariciones de un determinado elemento en la lista o tupla

```
cantidades = (4, 2, 5, 6, 8, 10, 3, 5, 7, 4, 2, 6)
cantidades.count(4)
```

2

### Sum

Suma todos los elementos de la lista

```
cantidades = (4, 2, 5, 6, 8, 10, 3, 5, 7, 4, 2, 6)
sum(cantidades)
```

62

# Métodos de las listas

## Sort

En el caso de las listas, no así para las tuplas, existe el método sort para reordenarlas. Al ser inmutables no se pueden modificar por eso las tuplas no lo admiten y las listas si.

```
cantidadesLista = list(cantidades)
cantidadesLista.sort()
print(cantidadesLista)
[2, 2, 3, 4, 4, 5, 5, 6, 6, 7, 8, 10]
```

```
cantidadesLista.sort(reverse=True)
print(cantidadesLista)
[10, 8, 7, 6, 6, 5, 5, 4, 4, 3, 2, 2]
```

## Append

Agregando un valor al final de una lista

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
print(listado)
listado.append("ALUA")
print(listado)
```

```
['GGAL', 'PAMP', 'YPFD', 'CEPU', 'EDN', 'LOMA', 'CRES'] ['GGAL',
'PAMP', 'YPFD', 'CEPU', 'EDN', 'LOMA', 'CRES', 'ALUA']
```

## Extend

Agregado de una lista entera a otra

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
listadoPanelGral = ["BHIP", "CELU", "CTIO"]
listado.extend(listadoPanelGral)
print(listado)
```

```
['GGAL', 'PAMP', 'YPFD', 'CEPU', 'EDN', 'LOMA', 'CRES', 'BHIP', 'CELU', 'CTIO']
```

## Insert

Agregado de un elemento en un lugar definido

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
listado.insert(3, "TXAR")
print(listado)
['GGAL', 'PAMP', 'YPFD', 'TXAR', 'CEPU', 'EDN', 'LOMA', 'CRES']
```

## Pop (muy interesante)

Elimina el elemento requerido y lo devuelve como return del método

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
eliminado = listado.pop(2)
print("El elemento eliminado es "+eliminado)
print("\nLa lista final es la siguiente")
print(listado)
```

El elemento eliminado es YPFD

La lista final es la siguiente

```
['GGAL', 'PAMP', 'CEPU', 'EDN', 'LOMA', 'CRES']
```

## Reverse

Da vuelta la lista, esto es interesante cuando nos viene una serie temporal en un orden y la queremos exactamente al revés

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
listado.reverse()
print(listado)

['CRES', 'LOMA', 'EDN', 'CEPU', 'YPFD', 'PAMP', 'GGAL']
```

## Clear

Borra todos los elementos de la lista

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
listado.clear()
print(listado)
```

[]

## Diccionarios

Un diccionario es una estructura Clave=>Valor

Por ejemplo, en lugar de tener un listado de las notas de mis alumnos, podría tener un diccionario con la nota asignada al nombre de cada uno

```
diccionario = {"ALUA":29.35, "BBAR":120.85, "BMA":265.2, "BYMA":295}
```

dict

```
print(diccionario)
```

```
{'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 295}
```

Dos Formas de Acceder a los valores de una clave de un diccionario

```
print(diccionario['BBAR'])
```

120.85

```
print(diccionario.get('BBAR'))
```

120.85

Supongamos que sabemos que podría no estar el ticker buscado en el diccionario

Para ello conviene usar el método get, ya que tiene un valor por default en caso que el índice buscado no se encuentre para evitar que tire error el programa, ese valor por default es **None** es decir un dato nulo pero como decía no me tira error

```
print(diccionario.get('FRAN'))
```

None

Asimismo, puedo elegir el valor por default que me devuelva en caso de no encontrar el índice en el diccionario, es el segundo argumento que no es obligatorio

```
print(diccionario.get('FRAN', 0))
```

0

### Accedemos por separado a las Claves y Valores

```
print(diccionario.keys())
dict_keys(['ALUA', 'BBAR', 'BMA', 'BYMA'])
```

```
values = diccionario.values()
print(values)
dict_values([29.35, 120.85, 265.2, 295])
```

Sí queremos tener los valores como una lista y no como un objeto de valores de diccionario:

```
values = list(values)
print(values)

[29.35, 120.85, 265.2, 295]
```

### Accedemos a los pares Clave => Valor

```
items = diccionario.items()
print(items)

dict_items([('ALUA', 29.35), ('BBAR', 120.85), ('BMA', 265.2), ('BYMA', 295)])
```

```
items = diccionario.items()
print(list(items)[0])

('ALUA', 29.35)
```

### Pop

Tomar un Elemento y borrarlo pasando la clave, funciona igual que en las listas, aquí devuelve solo el valor, no la clave

```
diccionario = {"ALUA":29.35,"BBAR":120.85,"BMA":265.2,"BYMA":295}
itemTomado = diccionario.pop("BBAR")
print(itemTomado)
print(diccionario)
```

```
120.85
{'ALUA': 29.35, 'BMA': 265.2, 'BYMA': 295}
```

## Update

Actualiza los valores de un diccionario según las coincidencias de las claves del segundo

```
diccionario1 = {"ALUA":29.35,"BBAR":120.85,"BMA":265.2,"BYMA":295}  
diccionario2 = {"BYMA":290,"CEPU":29,"COME":3,"CRES":40.7}  
diccionario1.update(diccionario2)  
print(diccionario1)
```

{'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29, 'COME': 3, 'CRES': 40.7}

## Insertar nuevo elemento a un diccionario

```
dic1 = {"ALUA":29.35,"BBAR":120.85,"BMA":265.2,"BYMA":295}  
dic1["CEPU"] = 29  
print(dic1)
```

{'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 295, 'CEPU': 29}

## Inicialización de Listas y Diccionarios

Si quisieramos agregar elementos a listas o diccionarios que no existen vamos a tener errores

```
dicNuevo["GGAL"] = 90
dicNuevo
```

```
NameError Traceback (most recent call last)
<ipython-input-142-a6426c4a9a09> in <module>
----> 1 dicNuevo["GGAL"] = 90
      2     dicNuevo
NameError: name 'dicNuevo' is not defined
```

Esto pasa porque al momento de asignarle el primer valor python no sabe a qué nos referimos porque nunca lo nombramos antes

Para evitar este error debemos definirlos al inicio como una lista o diccionario vacío según corresponda

```
dicNuevo = {}
dicNuevo["GGAL"] = 90
dicNuevo
```

```
{'GGAL': 90}
```

```
listaNueva = []
listaNueva.append("GGAL")
listaNueva
```

```
['GGAL']
```

## Asignación múltiple

Ya veremos otras formas de generar varias listas o diccionarios vacíos en una sola línea, pero por ahora podemos usar este método

La asignación múltiple está permitida en python y es básicamente asignarles a muchas variables el mismo valor

```
lista1 = lista2 = lista3 = []
print(lista1, lista2, lista3)
```

```
[] [] []
```

# Ejercicios

Dada la siguiente lista

```
listado = [1,2,3,4,5,6,7,8,9,10,11,12]
```

1 - Los números pares

2 - Los valores del 4 hasta el final de la lista

3 - Los valores del 5 hasta el 10

4 - Los valores menores a 10 que sean pares

5 - los múltiplos de 3 ordenados al revés

6 - El promedio de todos los números (usando las funciones sum y len)

Dadas las siguientes listas:

```
líderes = ["GGAL","PAMP","YPFD","TECO2","EDN","LOMA","BBAR"]
```

```
galpones = ["AGRO","FIPL","MIRG","GARO","LONG"]
```

7 - Elegir un galpón al azar y agregarlo a la lista líderes al final

8 - Elegir un galpón al azar y remplazar a EDN por ese galpón

9 - Elegir 3 galpones al azar y remplazar una líder cualquiera por esos dos galpones

Dado el siguiente diccionario:

```
panel = {'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29, 'COME': 3, 'CRES': 40.7}
```

10 - Remplazar el precio de BBAR por un precio al azar que elija python que varie +/- \$1 del precio que tiene (con precisión de 0,01)

11 - Remplazar el precio de BBAR por un precio al azar que elija python que varie +/- 3% del precio que tiene (con precisión de 0,01)

12 - Elegir un ticker al azar y mostrarlo

13 - Re-ordenar el diccionario al azar

14 - Re-ordenar el diccionario alfabéticamente en forma inversa (de la Z a la A)

## Respuestas

```
#-----#
# Rta Ejercicio 1 #
# -----#
listado = [1,2,3,4,5,6,7,8,9,10,11,12]
listado[1::2]
[2, 4, 6, 8, 10, 12]
```

```
#-----#
# Rta Ejercicio 2 #
# -----#
listado = [1,2,3,4,5,6,7,8,9,10,11,12]
listado[3:]
[4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
#-----#
# Rta Ejercicio 3 #
#-----#
listado = [1,2,3,4,5,6,7,8,9,10,11,12]
listado[4:10]
[5, 6, 7, 8, 9, 10]
```

```
#-----#
# Rta Ejercicio 4 #
#-----#
listado = [1,2,3,4,5,6,7,8,9,10,11,12]
listado[1:-3:2]
[2, 4, 6, 8]
```

```
#-----#
# Rta Ejercicio 5 #
#-----#
listado = [1,2,3,4,5,6,7,8,9,10,11,12]
listado[-1::-3]
[12, 9, 6, 3]
```

```
#-----#
# Rta Ejercicio 6 #
#-----#
listado = [1,2,3,4,5,6,7,8,9,10,11,12]
sum(listado)/len(listado)
```

6.5

```
#-----#
# Rta Ejercicio 7 #
#-----#
import random
lideres = ["GGAL", "PAMP", "YPFD", "TECO2", "EDN", "LOMA", "BBAR"]
galpones = ["AGRO", "FIPL", "MIRG", "GARO", "LONG"] galpon =
random.choice(galpones)
lideres.append(galpon)
lideres
['GGAL', 'PAMP', 'YPFD', 'TECO2', 'EDN', 'LOMA', 'BBAR', 'LONG']
```

```
#-----#
# Rta Ejercicio 8 #
#-----#
lideres = ["GGAL", "PAMP", "YPFD", "TECO2", "EDN", "LOMA", "BBAR"]
galpones = ["AGRO", "FIPL", "MIRG", "GARO", "LONG"] galpon =
random.choice(galpones)
lideres[4] = galpon
lideres
['GGAL', 'PAMP', 'YPFD', 'TECO2', 'LONG', 'LOMA', 'BBAR']
```

```
#-----#
# Rta Ejercicio 9 #
#-----#
lideres = ["GGAL", "PAMP", "YPFD", "TECO2", "EDN", "LOMA", "BBAR"]
galpones = ["AGRO", "FIPL", "MIRG", "GARO", "LONG"] galpon_x3 =
random.sample(galpones, 3)
indice = random.randrange(0,5)
lideres[indice] = galpon_x3
lideres
['GGAL', 'PAMP', 'YPFD', ['LONG', 'GARO', 'AGRO'], 'EDN', 'LOMA', 'BBAR']
```

```

#-----#
# Rta Ejercicio 10 #
#-----#
import random
panel = {'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29}
valorAzar = random.uniform(-1,1)
panel['BBAR'] = round(panel['BBAR'] + valorAzar,2)
panel
{'ALUA': 29.35, 'BBAR': 121.55, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29}

```

```

#-----#
# Rta Ejercicio 11 #
#-----#
import random
panel = {'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29}
banda = 0.03 * panel['BBAR']
valorAzar = random.uniform(-banda,banda)
panel['BBAR'] = round(panel['BBAR'] + valorAzar,2)
panel
{'ALUA': 29.35, 'BBAR': 117.37, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29}

```

```

#-----#
# Rta Ejercicio 12 #
#-----#
import random
panel = {'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29}
tickers = panel.keys()
random.choice(list(tickers))

```

'BYMA'

```

#-----#
# Rta Ejercicio 13 #
#-----#
panel = {'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29}
keys = list(panel.keys())
random.shuffle(keys)
listaReordenada = [(key, panel[key]) for key in keys]
dict(listaReordenada)
{'BYMA': 290, 'BBAR': 120.85, 'ALUA': 29.35, 'CEPU': 29, 'BMA': 265.2}

```

```
#-----#
# Rta Ejercicio 14 #
#-----#
panel = {'ALUA': 29.35, 'BBAR': 120.85, 'BMA': 265.2, 'BYMA': 290, 'CEPU': 29}
keys = list(panel.keys())
keys.sort()
keys.reverse()
listaReordenada = [(key, panel[key]) for key in keys]
dict(listaReordenada)
{'CEPU': 29, 'BYMA': 290, 'BMA': 265.2, 'BBAR': 120.85, 'ALUA': 29.35}
```

## Decisiones

Las expresiones de comparación siempre devuelven True o False, no hay más posibilidades, y esa devolución dependerá de lo que comparemos

Operadores Relacionales:

- IGUAL ==
- DISTINTO !=
- Mayor >
- Mayor o igual >=
- Menor <
- Menor o igual <=

```
COME = 3  
GGAL = 100
```

Una vez definidas las variables "a" y "b" vamos a ver que devuelven las expresiones de comparación

```
COME > GGAL
```

False

```
COME <= GGAL
```

True

Ojo con cómo se pregunta por una igualdad, es muy común que nos pase de querer comparar si "a" es igual a "b" y escribirlo con un solo signo igual.

Pero ¿qué pasa si preguntamos eso poniendo un solo signo igual?

Lo que pasaría en ese caso es que a la variable **a** le estaríamos asignando el valor que tenga la variable **b**

```
COME == GGAL
```

False

```
COME != GGAL
```

True

¿Y qué pasa si comparamos un número con un texto? apuesto que siempre va a tirar error...

```
COME > "COME"
```

-----  
TypeError

```
Traceback (most recent call last) <ipython>
input-11-9ae4ff098aa0> in <module>
----> 1 COME > "COME"
TypeError: '>' not supported between instances of 'int' and 'str'
```

¿Pero siempre da error comparar un número con un string?

```
COME != "COME"
```

True

mira vos? y sí. le estoy preguntando si son distintos y efectivamente son distintos.

```
COME == "COME"
```

False

¿Y si comparamos un número con un booleano?

```
COME > True
```

True

aaaa, bueno, sorpresa? el tema es que True es como sinónimo de 1 y False como sinónimo de 0 Es decir que en esa comparación está diciendo que  $3 > 1$  por eso me devuelve verdadero

```
COME == True
```

False

```
1 == True
```

True

```
1 <= True
```

True

```
0 == False
```

True

ojo con esto, este tipo de cosas suele generar ese tipo de bugs o fallas muy difíciles de encontrar

## ¿Y qué pasa con los valores nulos en las comparaciones?

```
COME == None
```

False

```
COME > None
```

```
TypeError  
Traceback (most recent call last):  
  File <ipython-input-42-52baf6bf6235>, in <module>  
    1 COME > None  
----> 2 TypeError: '>' not supported between instances of 'int' and 'NoneType'
```

Ojo con esto que puede traer a confusión, si bien el valor nulo "None" es evaluado como falso (ya lo veremos en unos minutos), si evaluó comparar una variable nula con el 0 me devuelve falso, porque Python no tiene comparación de igualdad estricta, por lo tanto, la igualdad es estricta de por si

```
0 == None
```

False

## Sentencia IF

Obviamente la sentencia IF lo que hace es preguntar si se cumple o no una condición

En caso que la condición preguntada sea **True** ejecuta la línea que sigue después de los DOS PUNTOS, caso negativo termina ahí si no hay **else**, pero si hubiera un **else**, ejecuta la línea o líneas que siguen después de los DOS PUNTOS del **else**

```
1 Estructura del IF
2
3 if (condicion):
4
5     1º línea a ejecutar si condición es verdadera
6     2º línea a ejecutar si condición es verdadera
7     .....
8     nº línea a ejecutar si condición es verdadera
9
10 else:
11
12     1º línea a ejecutar si condición es falsa
13     2º línea a ejecutar si condición es falsa
14     .....
15     nº línea a ejecutar si condición es falsa
16
```

## Concepto de la indentación

Como habrán observado, las líneas no están ordenadas todas hacia la izquierda, sino que dentro de cada "BLOQUE" de código hay como una especie de sangría.

Esa sangría no es sólo estética, aunque también ayuda a la lectura humana del bloque, sino que cumple una función que es indicarnos que instrucción o conjunto de instrucciones están dentro de cada bloque y que instrucción no.

Llamamos indentación a esa especie de sangría o espacio a la izquierda del texto que en programación nos sirve para dar jerarquía al orden en que el programa debe leer cada línea

En python es super fundamental el concepto de indentación porque de no respetarlo el intérprete nos va a dar un error de sintaxis, por ejemplo, fíjense en el ejemplo de estructura del IF, si ven todas las líneas que se ejecutan cuando la condición es verdadera, notarán que están todas indentadas al mismo nivel, es como una manera de agrupar todas estas líneas, en otros lenguajes como javascript o php no importa la indentación, pero se delimitan los bloques con llaves por ejemplo

Lo importante es siempre recordar al salir del **else**, que si no respetamos la indentación y ponemos una línea que debe ejecutarse dentro del **else** sin la indentación esa línea se ejecuta siempre ya que es la primera línea luego de terminar la sentencia **if**, ya sea que haya entrado por el **if** o por el **else**

Veamos algún ejemplo básico

```
if True:
    print("-Entra por verdadero")
    print("-Segunda linea a ejecutarse por haber entrado por verdadero")
else:
    print("-Entra por falso")
print("-Esta linea la imprime siempre, esta indentada en el mismo nivel que el IF")
```

-Entra por verdadero

-Segunda linea a ejecutarse por haber entrado por verdadero

-Esta linea la imprime siempre esta indentada en el mismo nivel que el IF

## Evaluación de True o False en distintos tipos de Variables

Veamos esto con algunos ejemplos

Cabe preguntarse ¿qué pasa si preguntamos por una variable por ejemplo Nula?

En el caso de Python las variables nulas las evalúa como Falsas

```
variableNula = None
if variableNula:
    print("Evalúa Verdadero")
else:
    print("Evalúa Falso")
```

Evalúa Falso

¿Y si preguntamos por una variable = 0?

Las variables iguales a 0 python también las evalúa como falsas

```
variableCero = 0
if variableCero:
    print("Evalúa Verdadero")
else:
    print("Evalúa Falso")
```

Evalúa Falso

### Otro tipo de comparaciones

Un tipo de comparación muy común es entre una variable y una constante, un operador de mayor o menor o mayor o igual o menor o igual

```
COME = 3
if COME < 4:
    print("Evalua Verdadero")
else:
    print("Evalua Falso")
```

Evalúa Verdadero

También podemos comparar entre dos variables predefinidas anteriormente:

```
GGAL = 12
stopLoss = 11
if (GGAL < stopLoss) :
    print("Vender ya")
else:
    print("Sigo Adentro")
```

Sigo Adentro

Es muy común simplificar preguntando solo si la variable es verdadera y definir la comparación antes:

```
GGAL = 12
stopLoss = GGAL < 11
if stopLoss :
    print("Vender ya")
else:
    print("Sigo Adentro")
```

Sigo Adentro

Ustedes dirán ¿cuál es la ventaja de esta segunda opción?

Imaginen que en lugar de un stopLoss fijo, tenemos un trailing StopLoss, que como saben se va modificando a cada rato, en ese caso tendríamos que meter adentro de la pregunta del IF toda la lógica del stopLoss que mientras más compleja sea más complicaría la lectura, sin embargo es más sencillo definir aparte la variable stopLoss a la cual incluso, como más adelante veremos se le puede asignar la salida de una función y en el IF solo preguntar si se activó o no

Tiene una segunda ventaja no menor, y es la performance, en general vamos a ver muchos ifs dentro de bucles o ciclos, ya lo veremos en breve, que se repiten varias veces, y si cada vez que preguntamos tenemos que "evaluar" si es verdadero o falso, estamos perdiendo milisegundos que pueden ser importantes en el agregado, así que siempre es buena práctica definir la evaluación antes en una variable booleana y preguntar simplemente por esa variable

O sea que SIEMPRE busquen la forma de definir las comparaciones antes y llegar al if con un true o false

## Otros operadores útiles

Un operador muy útil para clarificar el código es el operador `is not` que significa obviamente "no es"

```
accion = "GGAL"
if accion is not "GGAL":
    print("No es GGAL")
else:
    print("Es GGAL")
```

Es GGAL

Otro operador extremadamente útil es el operador `in` que me va a servir para saber si un valor está o no dentro de una lista

```
accion = "YPF"
cartera = ["AAPL", "AMZN", "FB"]
if accion in cartera:
    print(F"{accion} está en la cartera")
else:
    print(F"{accion} NO está en la cartera")
```

YPF NO está en la cartera

También se puede usar el operador con el `not` adelante como cualquier otro operador

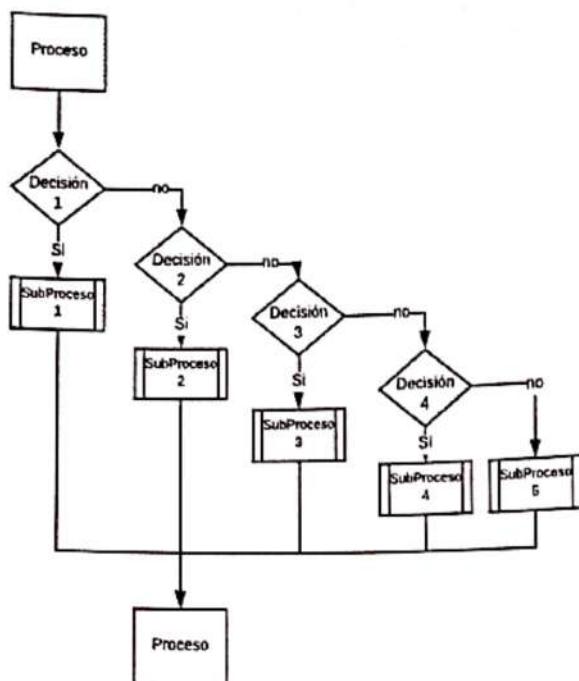
```
accion = "GGAL"
cartera = ["AAPL", "AMZN", "FB"]
if accion not in cartera:
    print(F"Ojo que ${accion} NO está en la cartera")
else:
    pass
```

Ojo que \$GGAL NO está en la cartera

Como verán la gran ventaja de usar este tipo de operadores es la claridad del código que se entiende por si solo

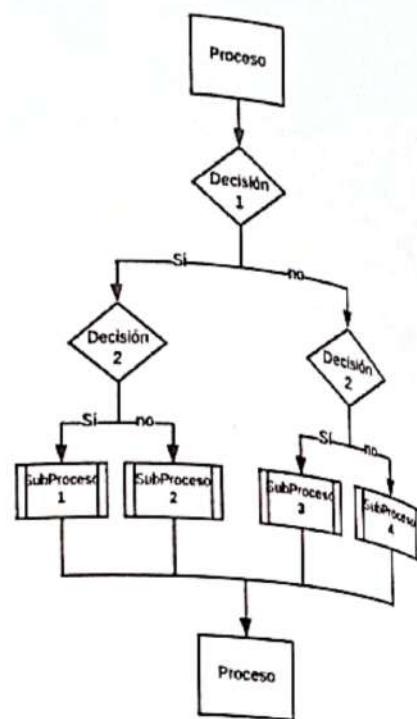
# Decisiones concatenadas y anidadas

## Decisiones Consecutivas o Concatenadas



Subprocesos = Decisiones + 1

## Decisiones Anidadas



Subprocesos = 2<sup>Decisiones</sup>

## Decisiones Consecutivas

Las decisiones consecutivas son muy útiles cuando tenemos que ir preguntando algo que en lugar de ser blanco/negro o si/no se va dividiendo como en grupos o escalas, por ejemplo, para rangos de malo, regular, bueno, muy bueno, excelente, vamos a evaluar una variable consecutivamente con los límites de esos grupos, veamos primero la estructura de las decisiones consecutivas:

```

if condicion_1:
    acciones si condicion_1 es verdaderas

elif condicion_2:
    acciones si condición_2 es verdadera

#...

elif condicion_n:
    acciones si condicion_n es verdadera

else:
    acciones si no fue verdadera ninguna de las condiciones anteriores
    sigue el programa
  
```

### Ejemplo típico de decisiones consecutivas

```
merval = 1800
if (merval>1500):
    print("All in Argy")
elif (merval>1000):
    print("Es por acá")
else:
    print("Piña historica")
```

All in Argy

## Decisiones Anidadas

Las decisiones anidadas son ni más ni menos que una decisión adentro de otra  
Veamos un ejemplo:

```
merval = 1200
usd = 60

if (merval>800):
    if (usd < 40):
        print("All in Argy")
    else:
        print("All in Brasil")
else:
    if (usd < 70):
        print("Piña historica")
    else:
        print("Yo te avisé")
```

All in Brasil

# Operadores Lógicos

## Tablas de VERDAD

Las tablas de verdad son solo esquemas de varios tipos de condiciones y operadores, por ejemplo, con el operador **and** el resultado de unir dos condiciones A y B será verdadero solo si ambas (AyB) son verdaderas

Por otro lado, si uno dos condiciones A o B con el operador **or**, su resultado será verdadero cuando ambas son verdaderas, pero también cuando una de ellas es verdadera y la otra no.

**Tabla de Verdad and**

A	and B	Resultado
F	F	F
F	V	F
V	F	F
V	V	V

**Tabla de Verdad or**

A	Or B	Resultado
F	F	F
F	V	V
V	F	V
V	V	V

En realidad las tablas de verdad no se usan nunca así como elemento teórico, como recurso pedagógico uno las muestra para que se entienda la lógica implícita en los operadores **and**, **or** y las comparaciones pero lo cierto es que uno se tiene que acostumbrar a "pensar como una máquina lógica" y asimilar estos conceptos de comparación binaria lo más humanamente posible para que nos sea natural

Veamos todo esto en unos ejemplos

Supongamos un caso muy típico, que tenemos un par de indicadores técnicos (partamos de la base de algo bien sencillo, dos medias móviles) si una media móvil rápida es mayor a una media móvil lenta digamos que es un indicador de compra y asociamos esa comparación a una variable "buySignal"

Luego vamos a suponer que para comprar necesitamos preguntar si hay liquidez suficiente en la cuenta, por lo tanto para gatillar una orden de compra vamos a tener que preguntar si el saldo de la cuenta es mayor a un determinado límite mínimo y si es afirmativo definir como True la variable liquidez

```
smaFast = 20.4
smaSlow = 20
price = 19
saldo = 15000

buySignal = smaFast > smaSlow
liquidez = saldo > 1000

buy = buySignal and liquidez
print(buy)
```

True

Ahora a nuestra lógica le vamos a agregar el estado "hold" que se va a mantener como True siempre que haya dado señal de compra y siempre y cuando el precio actual no sea menor a mi stopLoss

También se puede usar el operador **not** que lo que hace es preguntar por lo opuesto

```
smaFast = 20.4
smaSlow = 20
price = 19
saldo = 15000
stopLossPrice = 18

buySignal = smaFast > smaSlow
liquidez = saldo > 1000
stopLoss = price <= stopLossPrice

hold = buySignal and liquidez and not
stopLoss print(hold)
```

True

# Manejo básico de errores, Try/Except

La sentencia Try/Except se usa para evitar que el script se corte inesperadamente por una cuenta o una instrucción que da error porque el intérprete no puede resolverla

Recordemos que el flujo de información en un script es continuo y cualquier error que el intérprete no sepa resolver va a cortar ese flujo y va a impedir que el script termine su rutina, por lo tanto, este método try/except nos va a permitir "intentar" una instrucción que sospechamos puede dar error, y si ocurre el error realizar la instrucción que le pasamos en el "except"

Es como advertirle al intérprete, "ojo que te mando esta instrucción dudosa, fíjate primero si no es para kilombo" y ahí el intérprete en lugar de seguir de una el flujo normal del script, lo que hace es ver si falla y si falla se va por el otro lado (el del except)

Veámoslo con el típico error de la división por cero, hay veces que no sabemos si nos va a venir un dato vacío o no, y ahí es útil usar este método:

```
volumen = 1200
dias = 0

try:
    volMedio = volumen / dias
except:
    volMedio = None

print(volMedio)
```

None

```
volumen = 1200
dias = 0
volMedio = None

try:
    volMedio = volumen / dias
except:
    pass

print(volMedio)
```

None

También existe la instrucción **pass** que lo que hace es seguir de largo

Y por último también existe la instrucción **finally** que lo que hace es ejecutarse independientemente de si salió por el **try** o por el **except**

Como verán en los ejemplos pasa siempre por el bloque **finally**, independientemente si haya habido o no un error

Probablemente se estarán preguntando para que sirve el bloque **finally** si podría haber puesto esas líneas (El print "por acá pasa siempre") al mismo nivel que el **except** al terminar ese bloque

Les cuento que yo me pregunté lo mismo cuando vi esto y tardé bastante en entender el por qué..

Es una cuestión mas bien de prolidad, es exactamente lo mismo poner un bloque de instrucciones en el **finally** o ponerlo después del bloque **except** sin mas,, Pero queda mas prolado y entendible cuando todo el bloque Try/Except/Finally tiene una lógica asociada, por ahora les parecerá una pavada, es normal que piensen eso, como les digo me pasó lo mismo, pero estos detalles a la larga hacen el código más limpio y mantenible

```
volumen = 1200
dias = 2

try:
    volMedio = volumen / dias
except:
    volMedio = None
finally:
    print("por acá pasa siempre")

print(volMedio)
```

por acá pasa siempre  
600.0

```
volumen = 1200
dias = 0

try:
    volMedio = volumen / dias
except:
    volMedio = None
finally:
    print("por acá pasa siempre")

print(volMedio)
```

por acá pasa siempre  
None

## Catching, capturando el tipo de error

Además de salir por el except evitando el corte del script por un error, también podemos "capturar" el error que salta en el try

Para ello debemos importar la librería sys que es una librería básica de python  
Y como verán en el bloque except capturamos el error usando el método exc\_info() de la librería sys que importamos antes.

```
import sys
volumen = 1200
dias = 0

try:
    volMedio = volumen / dias
except:
    error = sys.exc_info()[1]
    print("Error detectado:",error)
```

Error detectado: división by zero

```
import sys
volumen = 1200
dias = 2

try:
    volMedio = volumen / dias
    print(volMedio)
except:
    error = sys.exc_info()[1]
    print("Error detectado:",error)
```

600.0

# Ejercicios

Partiendo de las siguientes variables:

- **price** que tiene el precio actual del activo en el mercado
- **buyPrice** que tiene el valor al que compramos el activo al ingresarla a la cartera
- **smaFast** y **smaSlow** que son dos medias móviles que usaremos para saber si está en tendencia

Ir modificando la variable **price** para verificar que el script anda Configurar las variables con los siguientes valores

- **smaFast = 100**
- **smaSlow = 98**
- **buyPrice = 106**

1- Armar un script que me defina en una sola condición la variable **hold** para un activo es verdadera o falsa, en función de las siguientes premisas:

- Que esté en tendencia bull, asumiendo para ello que la media rápida sea un 2% mayor a una media lenta
- Que no rompa un stopLoss del 5% abajo del precio de compra
- Que la ganancia acumulada no sea superior al 20% ya que en ese caso se tomaría ganancias

Y verificar que nos devuelve lo siguiente para los siguientes precios:

- **price = 97 => False**
- **price = 100 => False**
- **price = 101 => True**
- **price = 121 => True**
- **price = 128 => False**

2 - Con las mismas premisas del ejercicio anterior, modificarlo para que me devuelva en una variable **state** un string que me indique si estoy invertido en ese activo, pero además en caso de ser falso me indique si se salió por stopLoss o por toma de ganancias

Verificar que nos devuelve lo siguiente para los siguientes precios:

- **price = 97 => Salida por stopLoss**
- **price = 100 => Salida por stopLoss**
- **price = 101 => Invertidos**
- **price = 121 => Invertidos**
- **price = 128 => Salida por takeProfit**

3 - Reforzar la lógica del ejercicio anterior agregando a la condición del stopLoss que el precio tiene que cortar a la media móvil lento para abajo para disparar el stopLoss. Verificar que nos devuelve lo siguiente para los siguientes precios:

- price = 97 => Salida por stopLoss
- price = 100 => Invertidos
- price = 101 => Invertidos
- price = 121 => Invertidos
- price = 128 => Salida por takeProfit

4 - Reforzar la lógica del ejercicio anterior, haciendo que para gatillar el take profit puede o bien darse el 20% de ganancia o bien un 10% de ganancia y el precio superar en más del 20% a la media móvil rápido Verificar que nos devuelve lo siguiente para los siguientes precios:

- price = 97 => Salida por stopLoss
- price = 100 => Invertidos
- price = 101 => Invertidos
- price = 121 => Salida por takeProfit
- price = 128 => Salida por takeProfit

5 - Hacer un script que arranque definiendo el diccionario:

```
cartera = {"AAPL":30,"AMZN":25,"NFLX":20,"FB":10,"KO":5,"USD":10}
```

Que representa los activos de la cartera y sus porcentajes de peso y que me pida el ticker por un input y me devuelva el % de cada uno y si ingreso en el input un activo inexistente que me devuelva un mensaje aclarando que ese activo no está en la cartera

Tener en cuenta que el script debe funcionar si el usuario ingresa el ticker con alguna o todas sus letras en minúscula

Dada la siguiente lista de activos

```
activos1 = ["ALUA", "BMA", "BYMA", "CEPU", "COME", "CRES", "CVH", "EDN", "GGAL", "MIRG"]  
activos2 = ["PAM", "SUPV", "TECO2", "TGNN04", "TGSU2", "TRAN", "TXAR", "VALO", "YPF"]
```

6- Armar un script que genere una lista llamada activos con todos los elementos de ambas listas y devolver la cantidad total

7- Pedir un ticker al usuario vía input () y devolver si está o no en la variable activos y que funcione independientemente si se ingresa el ticker en minúscula o mayúscula

8- Dado el siguiente diccionario

```
precios = {"GGAL": 80, "YPF": 500}
```

Hacer un script que le pida por un input () un ticker al usuario y

- Si el ticker está en el listado de precios le devuelva el precio
- Si no está el precio, pero sí se encuentra en el listado de activos, que le devuelva el mensaje que no tiene el precio de ese ticker (y nombrarlo en el mensaje)
- Si ni siquiera encuentra el ticker en el listado de activos que le avise que el ticker no está ni en el listado de tickers

9- Hacer un script que tome como dato la variable maxPerdida que sea un valor porcentual que representa el stopLoss máximo tolerado, y que devuelva en otra variable del tipo string llamada tipo las siguientes opciones:

- Stop Largo: Cuando  $8\% < \text{maxPerdida} \leq 15\%$
- Stop SemiLargo: Cuando  $6\% < \text{maxPerdida} \leq 8\%$
- Stop Standar: Cuando  $3\% < \text{maxPerdida} \leq 6\%$
- Stop Corto: Cuando  $1\% < \text{maxPerdida} \leq 3\%$
- Fuera de rango: Resto de los casos

## Respuestas

```
#-----#
# Rta Ejercicio 1 #
# -----#
smaFast = 100
smaSlow = 98
buyPrice = 106
price = 128
trend = smaFast > smaSlow * 1.02
stopLoss = price < buyPrice * 0.95
takeProfit = price > 1.2 * buyPrice
hold = trend and not stopLoss and not
takeProfit hold
```

False

```
#-----#
# Rta Ejercicio 2 #
# -----#
smaFast = 100
smaSlow = 98
buyPrice = 106
price = 128
trend = smaFast > smaSlow * 1.02
stopLoss = price < buyPrice * 0.95
takeProfit = price > 1.2 * buyPrice
hold = trend and not stopLoss and not
takeProfit if hold:
    state = "Estamos invertidos aun"
else:
    if stopLoss:
        state = "Salimos por Stop Loss"
    else:
        state = "Salimos por toma de ganancias"
print(state)
```

Salimos por toma de ganancias

```

#-----#
# Rta Ejercicio 3 #
#-----#
smaFast = 100
smaSlow = 98
buyPrice = 106
price = 128
trend = smaFast > smaSlow * 1.02
stopLoss = price < buyPrice * 0.95 and price < smaSlow
takeProfit = price > 1.2 * buyPrice
hold = trend and not stopLoss and not
takeProfit if hold:
    state = "Estamos invertidos aun"
else:
    if stopLoss:
        state = "Salimos por Stop Loss"
    else:
        state = "Salimos por toma de ganancias"
print(state)

```

Salimos por toma de ganancias

```

#-----#
# Rta Ejercicio 4 #
#-----#
smaFast = 100
smaSlow = 98
buyPrice = 106
price = 121
trend = smaFast > smaSlow * 1.02
stopLoss = price < buyPrice * 0.95 and price < smaSlow
takeProfit = price > 1.2*buyPrice or (price > 1.1*buyPrice and price > 1.2 * smaFast)
hold = trend and not stopLoss and not
takeProfit if hold:
    state = "Estamos invertidos aun"
else:
    if stopLoss:
        state = "Salimos por Stop Loss"
    else:
        state = "Salimos por toma de ganancias"
print(state)

```

Salimos por toma de ganancias

```

#-----#
# Rta Ejercicio 5 #
# -----#
cartera={"AAPL":30,"AMZN":25,"NFLX":20,"FB":10,"KO":5,"USD":10}
ticker=input("Ingrese el activo que quiere saber su % de ponderacion: ").upper()
try:
    print(f"La ponderación de {ticker} en su cartera es {cartera[ticker]} %")
except:
    print("Ingresaste cualquier verduda man")

```

Ingrese el activo que quiere saber su % de ponderación:  
aapl La ponderación de AAPL en su cartera es 30 %

```

#-----#
# Rta Ejercicio 6 #
# -----#
activos1=["ALUA","BMA","BYMA","CEPU","COME","CRES","CVH","EDN","GGAL","MIRG"]
activos2=["PAM","SUPV","TECO2","TGNN04","TGSU2","TRAN","TXAR","VALO","YPF"]

activos = activos1
activos.extend(activos2)
len(activos)

```

19

```

#-----#
# Rta Ejercicio 7 #
# -----#
activos1=["ALUA","BMA","BYMA","CEPU","COME","CRES","CVH","EDN","GGAL","MIRG"]
activos2=["PAM","SUPV","TECO2","TGNN04","TGSU2","TRAN","TXAR","VALO","YPF"]
activos = activos1

ticker=input("Ingrese un ticker: ").upper()
if ticker in activos:
    print("Si, el ticker está en el listado")
else:
    print("NO, el ticker no se encuentra")

```

Ingrese un ticker: alua  
Si, el ticker está en el listado

```

#-----#
# Rta Ejercicio 8 #
#-----#
precios = {"GGAL":80,"YPF":500}
activos1=["ALUA","BMA","BYMA","CEPU","COME","CRES","CVH","EDN","GGAL","MIRG"]
activos2=["PAM","SUPV","TECO2","TGNN04","TGSU2","TRAN","TXAR","VALO","YPF"]
activos = activos1 + activos2
tickerPrecios = precios.keys()

ticker=input("Ingrese un ticker: ").upper()
if ticker in tickerPrecios:
    print(F"El precio de {ticker} es {precios[ticker]}")
else:
    if ticker in activos:
        print(F"No tengo el precio de {ticker}")
    else:
        print(F"El ticker {ticker} ni siquiera está en mi listado de activos")

Ingrese un ticker: alua
No tengo el precio de ALUA

```

```

#-----#
# Rta Ejercicio 9 #
#-----#
# Alternativa 1

maxPerdida = 16

if maxPerdida > 1 and maxPerdida <=15:
    if maxPerdida > 8 :
        tipo = "Stop Largo"
    elif maxPerdida >6 :
        tipo = "Stop SemiLargo"
    elif maxPerdida >3 :
        tipo = "Stop Standar"
    elif maxPerdida >1 :
        tipo = "Stop Corto"
    else:
        tipo = "Fuera de rango"
tipo

```

'Fuera de rango'

```
#-----#
# Rta Ejercicio 9 #
# ----- #
# Alternativa 2

maxPerdida = 0

if maxPerdida > 8 and maxPerdida
    <=15: tipo = "Stop Largo"
elif maxPerdida >6 and maxPerdida
    <=15: tipo = "Stop SemiLargo"
elif maxPerdida >3 and maxPerdida
    <=15: tipo = "Stop Standar"
elif maxPerdida >1 and maxPerdida
    <=15: tipo = "Stop Corto"
else:
    tipo = "Fuera de rango"
tipo

'Fuera de rango'
```

# Archivos necesarios

En esta unidad utilizaremos algunos archivos para trabajar ellos son:  
 SPY.csv  
 balances.csv  
 estado\_resultados.csv

Todos ellos pueden descargarse de: [www.bit.ly/39Am64T](http://www.bit.ly/39Am64T)

## Ciclos Finitos e Infinitos

Los distintos tipos de esquemas de ciclos se pueden agrupar en:

- Ciclos Definidos: Sabemos de antemano exactamente cuántas veces se va a ejecutar (por ejemplo 5 veces)
- Ciclos Indefinidos: No sabemos cuántas veces se va a ejecutar, pero sabemos que tiene una cantidad máxima dada por el elemento a iterar
- Ciclos Infinitos
  - Ciclos Infinitos Interactivos: Se ejecuta cada vez que aparece un nuevo feed al ciclo, corta cuando se corta el feed
  - Ciclos Infinitos con centinela: De antemano se ejecuta infinitas veces hasta que aparece señal de corte

## Ciclos Definidos

Estos son los más sencillos, básicamente se utiliza el comando FOR para esto. En estos ciclos sabemos de antemano la cantidad de iteraciones que hará

La estructura del comando FOR es muy sencilla, ya que se recorre un elemento iterable (generalmente una lista de elementos) y por cada elemento recorrido se ejecuta el cuerpo del FOR

```
listado = ["GGAL", "PAMP", "YPFD", "CEPU", "EDN", "LOMA", "CRES"]
for ticker in listado:
    print(ticker)
```

GGAL  
 PAMP  
 YPFD  
 CEPU  
 EDN  
 LOMA  
 CRES

Ojo que por ejemplo un string es decir un texto cualquiera es también un objeto iterable, ya que cada letra de la palabra se considera un elemento del string

Esto nos puede ser útil en algún momento para evaluar por separado cada elemento de un ticker de opciones

```
ticker="GFGC140.AB"
for letra in ticker:
    print(letra)
```

G  
F  
G  
C  
1  
4  
0  
.A  
B

## Iterando una lista

```
listado = ["AAPL","AMZN","NFLX","FB"]
for elemento in listado:
    print(elemento)
```

AAPL  
AMZN  
NFLX  
FB

## Iterando una Tupla

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

AAPL  
AMZN  
NFLX  
FB

## Iterando una lista de tuplas

```
pares = [("AAPL", "AMZN"), ("NFLX", "FB")]
for par in pares:
    print(f"{par[0]} es par con {par[1]}")
```

AAPL es par con AMZN  
NFLX es par con FB

## Iterando un Diccionario

```
cartera = {"AAPL": 240, "AMZN": 1700, "NFLX": 298, "FB": 146}
for elemento in cartera:
    print(f"{elemento} $ {cartera[elemento]})
```

AAPL \$ 240  
AMZN \$ 1700  
NFLX \$ 298  
FB \$ 146

## Ciclos definidos usando la función range

La función **range ()** admite 3 argumentos

- Desde
- Hasta
- incremento

El orden de los argumentos de la función range es así **range (desde, hasta, incremento)** Veamos ejemplos

```
for i in range(3):
    print(i)
```

0  
1  
2

Veámoslo con un listado predefinido

```
listado = ["AAPL", "AMZN", "NFLX", "FB", "GOOGL", "TSLA"]
for i in range(3):
    print(listado[i])
```

AAPL  
AMZN  
NFLX

Usemos un rango (desde, hasta)

```
for i in range(2,5):
    print(i)
```

2  
3  
4

Usamos un rango (desde, hasta, intervalo)

```
for i in range(4,10,3):
    print(i)
```

4  
7

¿Y para que nos puede servir esto?

Supongamos que tenemos un listado de precios cada 1 minuto, pero nos interesa ver en pantalla los cierres cada 3 minutos, en ese caso tendríamos algo similar a esto

```
precios = [112,113,111,114,110,109,110,111,112,115,112]
for i in range(0,len(precios),3):
    print(precios[i])
```

112  
114  
110  
115

## Creando listas con ciclos definidos

Como dije desde un comienzo siempre hay varias maneras de escribir el mismo código q en realidad derivan de varias maneras de pensar la solución

Vamos a hacer un script para llenar una lista con los elementos de la tabla de multiplicar de cualquier número "n" dado

En este primer ejemplo defino la tabla como una lista la cual va a estar compuesta por los elementos resultantes del recorrido de un ciclo FOR definido

```
n = 7
tabla = list(i*n for i in range(1,11))
tabla
```

[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]

```
n = 7
tabla = [i*n for i in range(1,11)]
tabla
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
```

Sin embargo, acá lo pensamos diferentes, primero definimos la tabla como una lista vacía, y luego creamos un ciclo FOR que lo que hará es ir llenando la tabla elemento por elemento

```
n = 7
tabla = []
for i in range(1,11):
    tabla.append(i*n)
tabla
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70]
```

## Ciclos Indefinidos Finitos

Son ciclos en los que a priori no sabemos cuántas veces se van a ejecutar. Para esto tenemos los comandos FOR y WHILE

En el caso del FOR el ciclo se ejecuta siempre para todos los elementos luego del IN así que vamos a tener que en algún momento indicarle dentro de un IF cuando queremos que deje de ejecutar, para ello vamos a usar el comando **BREAK**

```
precios = [112,113,111,114,110,109,110,111,112,115,112]
stopLoss = 110
for precio in precios:
    if(precio > stopLoss):
        print("Hold $" + str(precio))
    else:
        print("--- StopLoss at $" + str(precio) + " ---")
        break
```

```
Hold $112
Hold $113
Hold $111
Hold $114
--- StopLoss at $110 ---
```

Si quisieramos replicar el mismo código usando la sentencia WHILE, la pregunta que definirá si seguimos ejecutando dentro del ciclo es lo que viene justo después de la sentencia WHILE

Es decir que no hace falta poner un IF, sin embargo, si queremos ejecutar alguna línea al salir del ciclo, podemos poner un ELSE a su vez como no necesariamente tenemos que iterar un listado como en el FOR, lo que hacemos es usar un índice y un incrementador para ir refiriéndonos a cada elemento de la lista

```
precios = [112,113,111,114,110,109,110,111,112,115,112]
stopLoss = 110
i=0
while precios[i] > stopLoss:
    print("Hold $" + str(precios[i]))
    i += 1
else:
    print("--- StopLoss at $" + str(precios[i]) + " ---")
```

```
Hold $112
Hold $113
Hold $111
Hold $114
--- StopLoss at $110 ---
```

## Ciclos Infinitos

En estos casos no hay un elemento inicial a iterar como un listado de precios dado o un listado de tickers o cualquier listado predefinido, sino que por ejemplo dependemos de recibir o no un data feed, es decir que el ciclo se ejecutará cada vez que ingresa un nuevo dato al programa, por ahora vamos a hacer el ejemplito con inputs de usuario para no complicarla, más adelante veremos ejemplos reales tomando data en tiempo real

```
stopLoss = 110
dato = float("inf")
while dato > stopLoss:
    dato = float(input("Nuevo precio recibido $"))
    print("Hold $" + str(dato))
else:
    print("--- StopLoss at $" + str(dato) + " ---")
```

```
Nuevo precio recibido $123
Hold $123.0
Nuevo precio recibido $111
Hold $111.0
Nuevo precio recibido $109
Hold $109.0
--- StopLoss at $109.0 ---
```

# Corte de ciclos infinitos con corte por TimeOut

Como se darán cuenta la función anterior puede no cortar nunca en la vida, porque si infinitamente recibe datos y esos datos recibidos son siempre mayores al stopLoss la función nunca cortará, si bien esto en nuestro objetivo podría tener sentido, en realidad tenemos que pensar que podría haber eventos inesperados y esto podría tornarse peligroso

Hay miles de maneras de cortar un algo por "cosas raras" una muy típica son cuelgues del sistema de feed de datos, es decir si pasa una determinada cantidad de tiempo que no recibimos datos, nuestro bot puede quedar operando a ciegas con datos viejos pensando que este todo ok y en realidad dejamos de recibir datos porque se colgó algo y el mercado está colapsando.

Una manera muy típica de evitar esto es con los típicos TIMEOUT, que detectan cuando pasa un determinado tiempo de ejecución de una función o script

```
import time
stopLoss = 110
dato = float("inf")
timeout = time.time() + 5
while dato > stopLoss:
    if time.time() > timeout:
        print("Ojota que se colgó todo")
        break
    else:
        timeout = time.time() + 5
        dato = float(input("Nuevo precio recibido $"))
        print("Hold $" + str(dato))
else:
    print("--- StopLoss at $" + str(dato) + " ---")
```

```
Nuevo precio
recibido $123 Hold
$123.0
Ojota que se colgó todo
```

## Corte de Ciclos infinitos con Centinela

El centinela es un valor particular (que lo definimos nosotros) o bien puede ser una "lógica de secuencia de valores" que harán que nuestro ciclo corte

Por ejemplo, si lo que esperamos es un precio, podríamos definir que si recibimos el valor 0 se corte inmediatamente el script ya que muy probablemente estemos en presencia de algún tipo de cuelgue

```
stopLoss = 110
dato = float("inf")
while dato > stopLoss:
    dato = float(input("Nuevo precio recibido $"))
    if dato != 0:
        print("Hold $" + str(dato))
    else:
        print("Se corta el programa por recibir dato centinela")
        break
else:
    print("--- StopLoss at $" + str(dato) + " ---")
```

```
Nuevo precio
recibido $123 Hold
$123.0
Nuevo precio recibido $0
Se corta el programa por recibir dato centinela
```

Otra manera de escribir exactamente lo mismo es usando la sentencia continúe, en este caso no queda muy razonable, pero habrá casos en los que el código sea más entendible usando esta sentencia

```
stopLoss = 110
dato = float("inf")
while dato > stopLoss:
    dato = float(input("Nuevo precio recibido $"))
    if dato == 0:
        print("Se corta el programa por recibir dato centinela")
        break
    else:
        continue
    print("Hold $" + str(dato))
else:
    print("--- StopLoss at $" + str(dato) + " ---")
```

```
Nuevo precio
recibido $123 Nuevo
precio recibido $0
Se corta el programa por recibir dato centinela
```

# Importación de datos de una planilla

## Que es un archivo CSV

Antes que nada, vamos a repasar brevemente que es un archivo CSV  
La sigla CSV significa "Valores separados por comas" en inglés obviamente por eso está al revés,  
bueno, es un estándar,

Ventajas:

- 1- Son estándares, o sea todos los programas son capaces de entenderlos
- 2- Son simplemente archivos de texto con una separación con lo cual son livianos
- 3- Los puedo leer sin tener ningún software pago instalado en mi máquina
- 4- Sirven para guardar tablas de datos (matrices, como en un Excel)
- 5- Es compatible con Excel, con Google sheets, y con todos los gestores de base de datos

## Librería CSV

Es una librería estándar de python

Esta librería como se podrán imaginar nos va a permitir abrir y leer (también escribir) archivos CSV Vamos a ir viendo estas funciones directamente de unos ejemplos

En primer lugar, vamos a leer un archivo "SPY.csv" que tiene las cotizaciones de los últimos 20 años del ETF que replica el S&P500

Vemos lo que hace el ejemplo línea por línea

- 1- Importamos la librería CSV que tiene un método para leer justamente este tipo de archivos
- 2- Usamos el método Reader de la librería csv, y pasamos de argumento el archivo y el delimitador
- 3- Creamos una lista vacía
- 4- Armamos un ciclo definido que recorra todas las filas del archivo
- 5- En cada procesamiento del ciclo insertamos la fila a la lista creada
- 6- Imprimimos las primeras 5 filas

Este sería el código:

```
import csv
data = csv.reader(open('SPY.csv'), delimiter=';')
lista = []
for fila in data:
    lista.append(fila)
lista[0:5]
```

[[['timestamp', 'open', 'high', 'low', 'close', 'adjusted\_close', 'volume'],
 ['19/3/2020', '239.25', '247.38', '232.22', '240.51', '240.51', '288124389'],
 ['18/3/2020', '236.25', '248.37', '228.02', '240', '240', '324845381'],
 ['17/3/2020', '245.04', '256.17', '237.07', '252.8', '252.8', '260566279'],
 ['16/3/2020', '241.18', '256.9', '237.36', '239.85', '239.85', '295019288']]

En el ejemplo indicamos un CSV con el separador ";" si no se indica nada el estándar es "," en nuestro caso usamos este ejemplo para mostrarlo ya que ambos son muy comunes

Probamos acceder a una sola fila

```
lista[1]
```

['19/3/2020', '239.25', '247.38', '232.22', '240.51', '240.51', '288124389']

Y también podemos probar acceder a un elemento único de una determinada fila, en este caso al 3º elemento de la 2º fila

Recordemos siempre que en las listas el primer elemento siempre es el 0

```
lista[1][2]
```

'247.38'

Ahora vamos a ver solo a modo de ejemplo, como imprimiremos en pantalla la fecha y el precio de cierre de, por ejemplo, los primeros 5 datos de la serie

```
primeros5=lista[1:6]
for ohlc in primeros5:
    print(ohlc[0] + " Cierre $" + str(ohlc[4]))
```

19/3/2020 Cierre \$240.51  
 18/3/2020 Cierre \$240  
 17/3/2020 Cierre \$252.8  
 16/3/2020 Cierre \$239.85  
 13/3/2020 Cierre \$269.32

## Guardamos una columna en una lista

Como se habrán dado cuenta esto de tener todas las filas guardadas como filas no nos servirá mucho porque cada fila tiene datos que no tienen nada que ver entre sí, por lo general va a ser más útil agrupar datos de columnas de este tipo de series, veamos cómo puedo hacer eso

Partiendo de la base que tengo en la variable **lista** todas las filas, les recuerdo el código para ello

```
import csv
data = csv.reader(open('SPY.csv'), delimiter=';')
lista = []
for fila in data:
    lista.append(fila)
```

Lo que vamos a hacer es recorrer esas filas e ir llenando una lista con solo los valores de las fechas en este caso

Para ello usamos las siguientes líneas:

- 1- Definimos la lista de fechas vacía
- 2- Luego armamos el ciclo definido
- 3- Dentro del bloque del ciclo agregamos a la lista vacía
- 4- Por último, imprimimos las primeras 5 fechas a ver si este todo ok

Cabe aclarar que vamos a recorrer desde 1 (y no desde 0) hasta len(lista), porque la fila 0 contiene el encabezado y solo queremos los valores

```
fechas = []
for i in range(1, len(lista)):
    fechas.append(lista[i][0])
fechas[0:5]
```

['19/3/2020', '18/3/2020', '17/3/2020', '16/3/2020', '13/3/2020']

La manera abreviada de llenar la lista es definirla y en el mismo paso, adentro de los corchetes, la lleno Para llenarla, ponemos el valor y luego la iteración que define al valor

```
fechas = [ lista[i][0] for i in range(1, len(lista))]
fechas[0:5]
```

['19/3/2020', '18/3/2020', '17/3/2020', '16/3/2020', '13/3/2020']

Pero se preguntarán por que usamos la lista de filas y no hicimos esto directamente cuando recorrimos el CSV original

No sé si se lo preguntaron, pero si fue así, perfecto, excelente pregunta

La realidad hice esos pasos porque me pareció más pedagógico, pero claramente podríamos directamente cuando leemos el CSV armar la o las columnas que necesitaremos, veamos cómo sería

```
import csv
data = csv.reader(open('SPY.csv'), delimiter=';')
cierres_aj = []
for fila in data:
    cierres_aj.append(fila[5])
cierres_aj[0:4]
```

```
['adjusted_close', '240.51', '240', '252.8']
```

Como verán la única diferencia es que en este caso estamos guardando también el encabezado de la columna, lo podríamos quitar luego con la función del ()

```
del(cierres_aj[0])
cierres_aj[0:4]
```

```
['240.51', '240', '252.8', '239.85']
```

Como ven la función del (), lo que hizo fue borrar el primer valor (el que tenía el encabezado) pero luego "reseteó" el lugar 0 ya que al pedirle que me devuelva los primeros 4 valores arranca también desde 0 pero ya sin el encabezado

Por último, les muestro aquí una forma más abreviada de obtener una determinada columna con solo 5 líneas en total desde el CVS

```
import csv
data = csv.reader(open('SPY.csv'), delimiter=';')
cierres_aj = [ fila[5] for fila in data ]
del(cierres_aj[0])
cierres_aj[0:4]
```

```
['240.51', '240', '252.8', '239.85']
```

Ojo, no es necesario que sepan usar todos los métodos para nada, yo acá trato de mostrarles el mayor abanico posible por cuestiones didácticas.

Usen el que sientan más cómodo de leer y de interpretar, lo más razonable es simplificar la vida y no andar recorriendo gratuitamente una lista de datos que no vamos a necesitar muchas veces, ni de armar variables provisorias como la variable data que habíamos armado llenando la memoria innecesariamente, pero ojo, que también puede que prefieran hacerlo porque crean que a futuro necesitarán esa data completa para obtener otra columna sin tener que volver a recorrer el CSV

Como dijimos al principio de todo con las premisas de la programación, todo dependerá de las necesidades puntuales y también por qué no, de su comodidad, a veces lo prioritario es que el código sea legible o que simplemente sean comandos fáciles de recordar su uso

## Generamos una columna nueva a partir de los datos dados o de las columnas previamente generadas

Claramente una de las principales funciones de un script va a ser calcular cosas nuevas a partir de datos dados

Por ejemplo, partiendo de la data de cierres calcular la columna de variaciones diarias

Como la serie de datos está ordenada del más reciente primero hasta el más antiguo al final, el % de variación lo calcularemos como el cociente de:  $cierre_{aj}[i] / cierre_{aj}[i+1]$

y luego usaremos el método float () para transformar el dato que me viene encomillado es decir como string a decimal, y también usaremos el método round para limitar la cantidad de decimales del resultado

```
import csv
data =
csv.reader(open('SPY.csv'), delimiter=';')
cierres_aj = [ fila[5] for fila in data ]
del(cierres_aj[0])

variaciones = []
for i in range(0, len(cierres_aj)):
    try:
        var = round( (float(cierres_aj[i])/float(cierres_aj[i+1]) -1)*100 , 2 )
        variaciones.append(var)
    except:
        pass

variaciones[0:4]
```

[0.21, -5.06, 5.4, -10.94]

En el ejemplo anterior usamos el método try/except para evitar dos tipos de errores:

- 1- El error de "Out of range" es decir cuando llegue al final de la serie que no me tire error por querer buscar el elemento siguiente (que no existe, obvio)
- 2- El error potencial de alguna división por cero en caso que tenga un 0 en algún cierre (por defecto de la data feed)

## De CSV a Diccionario

En realidad, no es muy útil usar diccionarios para calcular parámetros y hacer análisis, el uso de diccionarios es más limitado a definiciones más bien estáticas, pero de todos modos de manera didáctica les dejo ejemplo de cómo armar un diccionario clave valor de fechas y cierres partiendo de un CSV

```
import csv
data = csv.reader(open('SPY.csv'), delimiter=';')
fechas_cierres = []
for fila in data:
    dicc = {}
    dicc[fila[0]] = fila[5]
    fechas_cierres.append(dicc)
fechas_cierres[0:4]
```

```
[{'timestamp': 'adjusted_close'},
{'19/3/2020': '240.51'},
{'18/3/2020': '240'},
{'17/3/2020': '252.8'}]
```

# Lectura de balances desde un CSV

Obvio que en realidad podemos leer cualquier tipo de dato que tengamos en un Excel o un CSV, en este caso los preparé de ejemplo un archivo con los 10 últimos balances trimestrales y anuales de los 4000 tickers más líquidos en USA que cotizan hace más de dos años.

Veamos primero que me trae en la primera fila

```
import csv  
data = csv.reader(open('balances.csv'), delimiter=';')  
filas = [ fila for fila in data ]  
filas[0]
```

```
['symbol', 'tipo', 'date', 'intangibleAssets', 'totalLiab', 'totalStockholderEquity',  
'otherCurrentLiab', 'totalAssets', 'commonStock', 'otherCurrentAssets',  
'retainedEarnings', 'otherLiab', 'otherAssets', 'totalCurrentLiabilities',  
'otherStockholderEquity', 'propertyPlantEquipment', 'totalCurrentAssets',  
'longTermInvestments', 'netTangibleAssets', 'shortTermInvestments', 'netReceivables',  
'longTermDebt', 'inventory', 'accountsPayable']
```

Como verán nos devuelve los encabezados de las columnas, veamos ahora un ejemplo para traer solo los blancos de FB

```
import csv  
data = csv.reader(open('balances.csv'), delimiter=';')  
balances = list()  
for fila in data:  
    if fila[0]=="FB":  
        balances.append(fila)  
print("La cantidad de balances es",len(balances),"El primero es", balances[0])
```

La cantidad de balances es 20

```
El primero es ['FB', 'trimestral', '31/12/2019', '894000000', '3232200000  
', '101054000000', '10854000000', '133376000000', 'NULL', '8000000', '556  
000000', '7327000000', '2759000000', '15053000000', '-489000000', '44783  
00000', '66225000000', 'NULL', '81445000000', '35776000000', '9518000000',  
'NULL', 'NULL', '1363000000']
```

Vamos ahora a mostrar solo el activo total (totalAssets) de los balances anuales (expresado en millones)

Para ello, lo que vamos a hacer es primero definir un diccionario vacío, y luego lo vamos a ir llenando (recuerden, clave=>valor) con las fechas como clave y la columna totalAssets como valor (la 8va columna, es decir la de índice 7 contando desde el 0)

```
activos_anuales = {}
for balance in balances:
    if balance[1]=="anual":
        activos_anuales[balance[2]] = round(int(balance[7]) /
1000000) activos_anuales

{
'31/12/2019': 133376,
'31/12/2018': 97334,
'31/12/2017': 84524,
'31/12/2016': 64961,
'31/12/2015': 49407,
'31/12/2014': 40184,
'31/12/2013': 17895,
'31/12/2012': 15103,
'31/12/2011': 6331,
'31/12/2010': 2990
}
```

Calculemos ahora para FB las ratios anuales de Activo Corriente sobre Pasivo Corriente

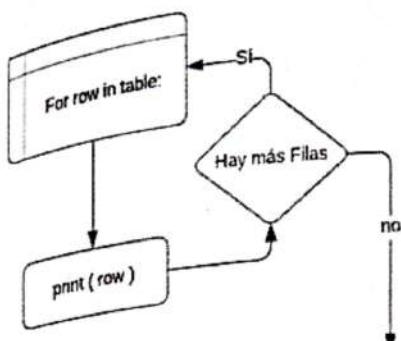
Ídem al procedimiento anterior solo que en lugar de guardar en el valor un dato, guardamos el cociente entre dos datos de la misma columna (el mismo balance)

```
ratio = {}
for balance in balances:
    if balance[1]=="anual":
        try:
            ratio[balance[2]] = round(int(balance[16]) / int(balance[13]),2)
        except:
            ratio[balance[2]] = "No se pudo calcular"
ratio

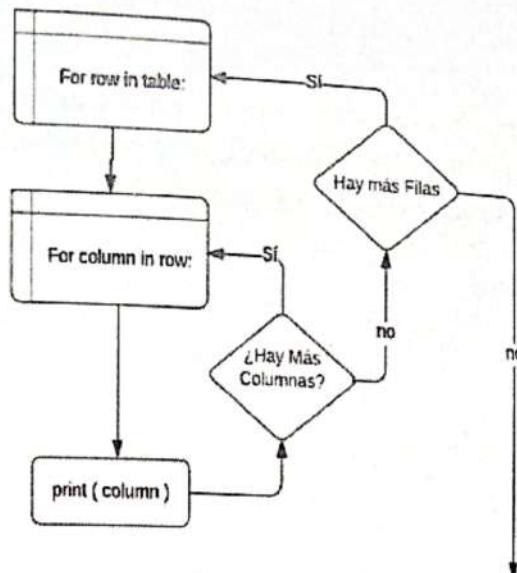
{
'31/12/2019': 4.4,
'31/12/2018': 7.19,
'31/12/2017': 12.92,
'31/12/2016': 11.97,
'31/12/2015': 11.25,
'31/12/2014': 9.4,
'31/12/2013': 11.88,
'31/12/2012': 10.71,
'31/12/2011': 'No se pudo calcular', '31/12/2010': 'No se pudo calcular'
}
```

# Bucles Anidados

Flujo del Ciclo FOR simple



Flujo del dos ciclos FOR anidados



El ejemplo básico que vemos en el diagrama de flujo de dos ciclos anidados es cuando queremos recorrer una tabla (filas y columnas) e imprimir por separado cada valor de la tabla

Para ello entramos a un primer ciclo FOR y recorremos las filas, y adentro del bloque del ciclo FOR entramos a otro ciclo FOR que recorra las columnas

```

table = [["f1c1","f1c2","f1c3"],["f2c1","f2c2","f2c3"]]
for row in table:
    for column in row:
        print(column)
  
```

```

f1c1
f1c2
f1c3
f2c1
f2c2
f2c3
  
```

## Ejemplos prácticos con bucles anidados

Bueno, ahora es donde se empieza a poner interesante el poder del scripting, si bien después vamos a ver librerías que nos facilitarán este tipo de recorrido de tablas, no siempre convendrá usarlas o no siempre las tendremos a disposición, por lo que siempre es conveniente saber hacerlo sin la ayuda de esas "librerías mágicas"

Tomando el archivo con los balances de las empresas, hagamos un diccionario con la ratio Activo Corriente / Pasivo Corriente, pero solo del último balance disponible (teniendo en cuenta que vienen ordenados de más reciente a más antiguo siempre) y de las siguientes empresas como para comparar la foto actual entre ellas

```
empresas = ["AAPL", "AMZN", "FB", "TSLA", "KO", "NFLX"]
```

Bueno, para esto tenemos que meter dos bucles anidados, en este caso vamos a recorrer primero la lista de empresas, y para cada empresa recorreremos las filas del CSV (podría hacerse al revés también)

Cuando recorramos las filas del CSV vamos a buscar para cada empresa el último balance y le calcularemos la ratio y lo pondremos en un diccionario clave=>valor donde la clave sea el ticker y el valor la ratio

```
import csv
data = csv.reader(open('balances.csv'), delimiter=',')
balances = [ fila for fila in data ]

empresas = ["AAPL", "AMZN", "FB", "TSLA", "KO",
            "NFLX"] screener = {}
for empresa in empresas:
    for balance in balances:
        if balance[0]==empresa and balance[1]=="anual":
            try:
                screener[empresa] = round(int(balance[16]) / int(balance[13]),2)
            except:
                screener[empresa] = "No se pudo calcular"
            break
print(screener)

{
'AAPL': 1.54,
'AMZN': 1.1,
'FB': 4.4,
'TSLA': 1.13,
'KO': 0.76,
'NFLX': 0.9
}
```

Bien, veamos un poco esto más en detalle, para cada empresa recorro todos los balances listados, después viene una pregunta con un IF y un AND, veamos

LINEA 9, el IF: lo que busco acá en cada fila es ver si la fila coincide con la empresa que estoy en ese punto del FOR, para ello pregunto por balance [0] es decir el primer valor de la fila balance, y justamente ese primer valor es el ticker o symbol como figura en el CSV. Bien, pero también pregunto si la siguiente columna de ese balance (balance [1]) es igual a "anual" ya que queremos ver la ratio solo en los balances anuales

Línea 10 el bloque Try/Except: Acá lo que hago, es calcular el ratio siempre que se pueda calcular y devolver un texto de que no se pudo llegar al caso pero que es lo de la linea 14?

Línea 14, break: Esta instrucción lo que hace es salirse del ciclo de menor jerarquía es decir en este caso el de los balances y no el de las empresas, y esto es necesario porque si no pongo esta instrucción va a seguir calculando el ratio con los balances subsiguientes y va a sobrescribir el valor, y recordemos que el ejercicio pedía calcular el ratio del último balance y nos decían que vienen ordenados del más reciente primero al más antiguo al final, así que solo lo tengo que calcular en la primera coincidencia y salirme de ese ciclo

## ¿Alguien pensó en la eficiencia de esa forma de resolver el ejercicio?

Si alguien se dio cuenta que eso que hicimos era sencillo de entender, pero poco eficiente a nivel esfuerzo computacional muy bien observado el punto

El razonamiento de la solución anterior es decir lo siguiente:

- 1- Agarro la primera empresa que quiero
- 2- Recorro toda la lista de balances hasta encontrar ESA empresa
- 3- Hago lo que tengo que hacer, calcular etc.
- 4- Sigo agarrando, la siguiente empresa de la lista y repito el punto 2 y 3

Ya de por sí es obvio que algo acá no es necesario, por sentido común, nosotros no pensamos así, diríamos vamos recorriendo los 63k balances y donde encontramos una empresa de la lista hacemos lo que tenemos que hacer y seguimos adelante, de esta forma recorreremos una sola vez los 63k balances y no una vez por cada empresa que necesite

El punto es que generalmente los ciclos anidados suelen ser muy poco eficientes, por lo general suele haber formas alternativas de resolver el tema sin necesidad de hacer ciclos anidados, de hecho, este es uno de esos casos, por lo general también no nos importa mucho la eficiencia del algoritmo en cuanto a su esfuerzo computacional, pensemos un poco este ejemplo

Mi CSV tenía 63k filas, y mi listado de empresas era de 6, es decir que la solución de los ciclos anidados hacia  $63k * 6 = 378k$  ciclos, porque para cada empresa volvía a recorrer otra vez los 63k balances. Si nuestro screener tenía 1000 empresas esa forma de resolverlo requeriría 63 millones de ciclos cuando se podría resolver en 63mil, es decir un algo 1000 veces más eficiente en "esfuerzo computacional"

Dicho todo eso entonces, resolvamos esto de una forma más sensata desde el punto de vista eficiencia

Para ello primero armo un diccionario con el nombre "calculado" en el que para cada empresa le asigno el booleano False

Este diccionario me va a servir para que una vez que empiezo a recorrer los balances si encuentro uno de una empresa listada, hago lo que tengo que hacer y le cambio ese booleano por True para indicar que esa empresa ya hice lo necesario y no me sobrescriba la ratio con los balances más viejos

```

import csv
data = csv.reader(open('balances.csv'), delimiter=',')
balances = [ fila for fila in data ]
empresas = ["AAPL", "AMZN", "FB", "TSLA", "KO",
"NFLX"] calculado = {}
for empresa in empresas:
    calculado[empresa] = False

screener = []
for balance in balances:
    empresa = balance[0]
    if empresa in empresas and balance[1]=="anual" and calculado[empresa] == False:
        try:
            screener[empresa] = round(int(balance[16]) / int(balance[13]),2)
        except:
            screener[empresa] = "No se pudo calcular"
        finally:
            calculado[empresa] = True
print(screener)

{
'AAPL': 1.54,
'AMZN': 1.1,
'FB': 4.4,
'KO': 0.76,
'NFLX': 0.9,
'TSLA': 1.13
}

```

## Archivos necesarios para los ejercicios

- balances.csv
- estado\_resultados.csv

Se pueden descargar de: <http://bit.ly/39Am64T> (<http://bit.ly/39Am64T>)

## Ejercicios

Dado el archivo SPY.csv que contiene en un CSV las cotizaciones de SPY hasta la fecha que fue generado:

1 - Generar un script que me devuelva la cantidad de subas diarias mayores al +5%, basados en los cierres ajustados

2 - Generar un script que arme un listado con los gaps de apertura (variación % del precio de apertura respecto al cierre anterior)  
Imprimir en pantalla los primeros 5 para verificar que estén correctos

3 - Basados en la lista de gaps anterior. Contabilizar del listado de gaps anterior la cantidad de gaps al alza, cantidad gaps a la baja y los neutros

4 - Contrastar los resultados de gaps al alza y gaps a la baja con la cantidad de ruedas que el intradiario (cierre sobre apertura) fue positivo vs los que fue negativo

5 - En función de las listas construidas en los ejercicios anteriores, armar un diccionario con los siguientes valores:

- El promedio de los gaps positivos y el promedio de los negativos
- El % de Gaps Positivos y Negativos sobre el total de Gaps
- El promedio de los intradiario positivos y el promedio de los negativos
- El % de Intras Positivos y Negativos sobre el total de Intras

- 6 - Tomando la lista de variaciones entre cierres ajustados, armar un diccionario con los siguientes valores:
- El promedio de las variaciones positivas y el promedio de las negativas
  - El % de variaciones Positivas, Negativas y neutras sobre el total
  - Inferir si el mayor movimiento del índice se da entre cierres, durante las ruedas o mientras el mercado está cerrado

Dado el archivo balances.csv y el archivo estado\_resultados.csv que contiene los últimos 10 estados de resultado anuales y trimestrales de las 4000 acciones más líquidas de USA:

- 7- Mostrar las columnas de la planilla
- 8- Mostrar los 4 últimos EBIT de AAPL trimestrales en millones de USD
- 9- Mostrar el último ratio costos/ingresos anuales de FB, AMZN, AAPL, NFLX y GOOGL
- 10- Mostrar el listado de empresas cuya ratio costos/ingresos anuales esté por debajo de 0.5, y que hayan invertido +5 B en R+D en el último ejercicio anual
- 11- Mostrar el promedio de inversión en R+D en Billones americanos de AAPL en los últimos 10 trimestres
- 12- Contar la cantidad de empresas que tuvieron el último balance anual una inversión en I+D de más de 1B americano
- 13- Calcular el valor promedio (en millones de dólares) que las empresas invirtieron en I+D el último balance anual
- 14- Elegir 5 empresas al azar de las que hayan invertido más de 400 millones de dólares

## Respuestas

```

#-----#
# Rta Ejercicio 1 #
#-----#
#-----#
import csv
data = csv.reader(open('SPY.csv'), delimiter=';')
lista = []
for fila in data:
    lista.append(fila)

cierres_aj = [ lista[i][5] for i in
range(1,len(lista)) ] variaciones = []
for i in range(0,len(cierres_aj)):
    try:
        var = round( (float(cierres_aj[i])/float(cierres_aj[i+1]) -1)*100, 2 )
    except:
        var = 0
    if var > 5:
        variaciones.append(var)
len(variaciones)

```

13

```

#-----#
# Rta Ejercicio 2 #
#-----#
#-----#
cierres = [ lista[i][4] for i in range(1,len(lista)) ]
aperturas = [ lista[i][1] for i in range(1,len(lista)) ]
gaps = []
for i in range(0,len(aperturas)):
    try:
        gap = round( (float(aperturas[i])/float(cierres[i+1]) -1)*100, 2 )
    except:
        gap = 0
    gaps.append(gap)
gaps[0:5]

```

[-0.31, -6.55, 2.16, -10.45, 6.04]

```

#-----#
# Rta Ejercicio 3 #
# ----- #

gaps_pos, gaps_neg, gaps_neutros = ([] for i in range(3))
for gap in gaps:
    if gap > 0 :
        gaps_pos.append(gap)
    elif gap < 0:
        gaps_neg.append(gap)
    else:
        gaps_neutros.append(gap)

print ( "Gaps Pos:", len(gaps_pos), " Gaps Neg:", len(gaps_neg), " Gaps Neutros:", len(gaps_neutros) )

```

Gaps Pos: 2672    Gaps Neg: 2270    Gaps Neutros: 90

```

#-----#
# Rta Ejercicio 4 #
# ----- #

intras, intras_pos, intras_neg, intras_neutros = ([] for i in range(4))

for i in range(0,len(aperturas)):
    intra = round( (float(cierres[i])/float(aperturas[i]) -1)*100, 2 )
    intras.append(intra)
    if intra > 0 :
        intras_pos.append(intra)
    elif intra < 0:
        intras_neg.append(intra)
    else:
        intras_neutros.append(intra)
print ( "Intras Pos:", len(intras_pos), "Intras Neg:",
len(intras_neg), "Intras Neutros:", len(intras_neutros) )

```

Intras Pos: 2640    Intras Neg: 2340    Intras Neutros: 52

```

#-----#
# Rta Ejercicio 5 #
#-----#
d = {}
d.update( {"GapsPos Media" : round(sum(gaps_pos)/len(gaps_pos),2) } )
d.update( {"GapsNeg Media" : round(sum(gaps_neg)/len(gaps_neg),2) } )
d.update( {"GapsPos %" : round( 100 * len(gaps_pos)/len(gaps) ,2 ) } )
d.update( {"GapsNeg %" : round( 100 * len(gaps_neg)/len(gaps) ,2 ) } )
d.update( {"GapsNeutros %" : round( 100 * len(gaps_neutros)/len(gaps) ,2 ) } )

d.update( {"IntraPos Media" : round(sum(intras_pos)/len(intras_pos),2) } )
d.update( {"IntraNeg Media" : round(sum(intras_neg)/len(intras_neg),2) } )
d.update( {"IntrasPos %" : round( 100 * len(intras_pos)/len(intras) ,2 ) } )
d.update( {"IntrasNeg %" : round( 100 * len(intras_neg)/len(intras) ,2 ) } )
d.update( {"IntrasNeutros %" : round( 100 * len(intras_neutros)/len(intras) ,2 ) } )

d
{'GapsPos Media': 0.42,
'GapsNeg Media': -0.46,
'GapsPos %': 53.1,
'GapsNeg %': 45.11,
'GapsNeutros %': 1.79,
'IntraPos Media': 0.63,
'IntraNeg Media': -0.71,
'IntrasPos %': 52.46,
'IntrasNeg %': 46.5,
'IntrasNeutros %': 1.03}

```

```

#-----#
# Rta Ejercicio 6 #
#-----#
vars_pos, vars_neg, vars_neutros = ([] for i in range(3))
for var in variaciones:
    if var > 0 :
        vars_pos.append(var)
    elif var < 0:
        vars_neg.append(var)
    else:
        vars_neutros.append(var)

dv = {}
dv.update({ "VarsPos Media" : round(sum(vars_pos)/len(vars_pos),2) })
dv.update({ "VarsNeg Media" : round(sum(vars_neg)/len(vars_neg),2) })
dv.update({ "VarsPos %" : round( 100 * len(vars_pos)/len(variaciones),2 ) })
dv.update({ "VarsNeg %" : round( 100 * len(vars_neg)/len(variaciones),2 ) })
dv.update({ "VarsNeutros %" : round( 100 * len(vars_neutros)/len(variaciones),2 ) })

dv
{'VarsPos Media': 0.76,
'VarsNeg Media': -0.86,
'VarsPos %': 54.01,
'VarsNeg %': 45.12,
'VarsNeutros %': 0.87}

```

```

#-----#
# Rta Ejercicio 7 #
#----- #

import csv
data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]
estados[0]

[
'symbol',
'tipo',
'date',
'researchDevelopment',
'effectOfAccountingCharges',
'incomeBeforeTax',
'minorityInterest',
'netIncome',
'sellingGeneralAdministrative',
'grossProfit',
'ebit',
'nonOperatingIncomeNetOther',
'operatingIncome',
'otherOperatingExpenses',
'interestExpense',
'extraordinaryItems',
'nonRecurring',
'otherItems',
'incomeTaxExpense',
'totalRevenue',
'totalOperatingExpenses',
'costOfRevenue',
'totalOtherIncomeExpenseNet',
'discontinuedOperations',
'netIncomeFromContinuingOps',
'netIncomeApplicableToCommonShares',
'preferredStockAndOtherAdjustments'
]

```

```

#-----#
# Rta Ejercicio 8 #
#----- #

import csv
data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]
q = 0
respuesta = []
for estado in estados:
    if estado[0]=="AAPL" and estado[1]=="trimestral" and q < 4 :
        respuesta.append(round(int(estado[10])/1000000))
        q = q+1
respuesta

```

[25569, 15625, 11544, 13415]

```

#----- #
# Rta Ejercicio 9 #
#----- #

import csv
data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]
empresas = [ "FB", "AMZN", "AAPL", "NFLX" , "GOOGL"]

calculado = {}
for empresa in empresas:
    calculado[empresa] = False

screener = {}

for estado in estados:
    empresa = estado[0]
    if empresa in empresas and estado[1]=="anual" and calculado[empresa] == False:
        try:
            screener[empresa] = round(int(estado[21])/int(estado[19]),2)
        except:
            screener[empresa] = "No se puede calcular"
        finally:
            calculado[empresa] = True

screener

```

```
{
'AAPL': 0.62,
'AMZN': 0.59,
'FB': 0.18,
'GOOGL': 0.44,
'NFLX': 0.62
}
```

```

#-----#
# Rta Ejercicio 10 #
#-----#

import csv
data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]

empresas = [estado[0] for estado in estados]

calculado = {}
for empresa in empresas:
    calculado[empresa] = False

screener = {}

for estado in estados:
    empresa = estado[0]
    if estado[1]=="anual" and calculado[empresa] == False :
        try:
            ratio = round(int(estado[21])/int(estado[19]),2)
            rd_bln = int(estado[3]) / 1000000000
            if ratio < 0.5 and rd_bln > 5:
                screener[empresa] = ratio
        except:
            ratio = "No se puede calcular"
        finally:
            calculado[empresa] == True
screener

```

```

{'ABBV': 0.24,
'AZN': 0.15,
'BMY': 0.28,
'BTI': 0.24,
'CELG': 0.08,
'CSCO': 0.33,
'DEO': 0.42,
'F': 0.4,
'FB': 0.14,
'GILD': 0.21,
'GOOG': 0.38,
'GOOGL': 0.3,
'GSK': 0.2,
'INTC': 0.35,
'JNJ': 0.31,
'LLY': 0.16,
'MRK': 0.4,
'MSFT': 0.2,
'NVS': 0.23,
'ORCL': 0.11,
'PFE': 0.24,
'QCOM': 0.4,
'SNY': 0.24,
'TEF': 0.48}

```

```

#-----#
# Rta Ejercicio 11 #
#-----#
#-----#
import csv
data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]
rd_lista = []
for estado in estados:
    empresa = estado[0]
    if estado[1] == "trimestral" and empresa == "AAPL" :
        try:
            rd_bln = int(estado[3]) / 1000000000
            rd_lista.append(rd_bln)
        except:
            pass
rd_promedio = sum(rd_lista)/len(rd_lista)
rd_promedio

```

3.7901

```

#-----#
# Rta Ejercicio 12 #
#-----#
#-----#
import csv
data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]
empresas = [estado[0] for estado in estados]

calculado = {}
for empresa in empresas:
    calculado[empresa] = False

listado = []

for estado in estados:
    empresa = estado[0]
    if estado[1] == "anual" and calculado[empresa] == False:
        try:
            rd_bln = int(estado[3]) / 1000000000
            calculado[empresa] = True
            if rd_bln > 1:
                listado.append(rd_bln)
        except:
            pass
len(listado)

```

```

#-----#
# Rta Ejercicio 13 #
#-----#

import csv
data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]
empresas = [estado[0] for estado in estados]
listado = []
calculado = {}
for empresa in empresas:
    calculado[empresa] = False

for estado in estados:
    empresa = estado[0]
    if estado[1]=="anual" and calculado[empresa] == False:
        try:
            rd_bln = int(estado[3]) / 1000000
            calculado[empresa] = True
            listado.append(rd_bln)
        except:
            pass
sum(listado)/len(listado)

```

181.6195679740683

```

#-----#
# Rta Ejercicio 14 #
#-----#

import csv
import random

data = csv.reader(open('estado_resultados.csv'), delimiter=';')
estados = [ fila for fila in data ]
empresas = [estado[0] for estado in estados]

calculado = {}
for empresa in empresas:
    calculado[empresa] = False

listado = []

for estado in estados:
    empresa = estado[0]
    if estado[1]=="anual" and calculado[empresa] == False:
        try:
            rd_bln = int(estado[3]) / 1000000
            calculado[empresa] = True
            if rd_bln > 400:
                listado.append(empresa)
        except:
            pass
random.sample(listado,10)

['MLNX', 'BIDU', 'ON', 'UTX', 'CRM', 'TEVA', 'TMO', 'SAP', 'BMY', 'NFLX']

```

# Títulos de esta colección

A mayo de 2020, los tres primeros tomos están ya editados y en proceso de impresión, la idea es un nuevo lanzamiento por mes hasta completar toda la colección (En principio de 16 tomos)

- t<sub>[0]</sub> Primeros Scripts: Introducción a Python y entornos de desarrollo recomendados. Estructuras de datos y variables. Estructuras lógicas básicas, ciclos y estructuras lógicas más complejas, concatenadas y anidadas
- t<sub>[1]</sub> DataFrames: Uso intensivo de la librería Pandas, estructuras de datos matriciales en formato de tablas, importación y exportación de datos con Excel y Python. Uso de funciones estadísticas con Pandas
- t<sub>[2]</sub> Matplotlib: Gráficas en Python, gráficas de líneas, gráficos financieros de velas, gráficas de columnas, columnas apiladas, columnas en 3D, anillos, gráficos estadísticos, histogramas y diagramas de cajas, introducción a modelización de distribuciones con scipy
- t<sub>[3]</sub> APIs de Market Data: Uso de funciones, JSON, requests http, APIs de datos de acciones, precios, fundamentales, indicadores, bonos, FX, opciones, futuros y datos económicos.
- t<sub>[4]</sub> APIs de Conexión a Mercados: APIs para acceso al mercado en Argentina API de Rofex. API de Invertir Online. API de Alpaca para inversiones en USA. Paper trading, sandbox o cuentas simuladas. APIs de Exchange cryptos para mercados spot, futuros y opciones. HitBTC, FTX, Deribit, Binance etc.
- t<sub>[5]</sub> SDKs y Librerías: SDKs de conexión a APIs financieras, librerías de dataFeed, de colocación de órdenes, de análisis de datos financieros y estadísticos (Análisis de tendencias, indicadores técnicos, análisis de portafolios, riesgo, etc)
- t<sub>[6]</sub> BBDD: Estructuras de bases de datos relacionales, uso de MySQL, sentencias, armado de funciones para CRUDs. Mantenimiento de Bases de datos automatizadas
- t<sub>[7]</sub> Backtestings: Ejecución offline y online de un backtesting. Guardado de datos en BBDD, consulta y actualización de backtestings. Métricas de un backtesting
- t<sub>[8]</sub> Screenings: Screenings en tiempo real y programados. Sistemas de alertas de screenings. Ejemplos de screenings de ratios de AF y de señales de AT. Screenings de ratios de análisis Quant.
- t<sub>[9]</sub> Portabilidad de estrategias: Estudio cruzado de estrategias backtesteadas en diferentes mercados, activos, grupos de activos, segmentación por afinidades, screenings y retesteos de portabilidad por diferentes agrupamientos

- t<sub>[10]</sub> Bots de trading: Ejemplos de diferentes estructuras de bots de trading, bots de balanceo de carteras y seguimiento de índices. Bots especulativos de scalping y swing trading. Bots de arbitrajes y arbitrajes estadísticos. Bots de coberturas de posiciones abiertas especulativa (reemplazo de stopLoss por cobertura con opciones). Ingeniería básica y costos operativos y de setup de un bot.
- t<sub>[11]</sub> Inteligencia artificial en la industria financiera. Aplicaciones a estrategias de inversión. Algoritmos de aprendizaje supervisado y no supervisado. Regresiones y clusterización. Correlación y predicción de valores y eventos
- t<sub>[12]</sub> Montecarlo y métodos modernos para medición de riesgos, simulación de eventos y posibilidades. Modelización de frontera eficiente de riesgo/retorno con simulación. Uso de las simulaciones seteo de escenarios
- t<sub>[13]</sub> Opciones a fondo con python, screenings con miles de activos con todas sus cadenas de opciones, pricing de primas por simulación y contraste contra modelos como el de Black&Scholes, griegas y algos de IA en regresiones de volatilidad para estrategias delta-neutral, long y short vega y vega-neutral.
- t<sub>[14]</sub> Ejemplos de implementaciones de bots reales con sistemas de control mixtos, gatillo de compra y de cierre de posición por leads de distintos tipos. Manejo dinámico de volumen de posición, sistemas activos y pasivos, bots de hedging, reporting real time etc
- t<sub>[15]</sub> Papers Quant y sus implementaciones en python

El presente libro tiene como objetivo hacer dar los primeros pasos en programación a gente que nunca antes programó. Tiene la particularidad de estar enfocado en temas de mercado de capitales ya que es parte de una serie de libros que van desde los primeros scripts hasta terminar en temas mas complejos como el análisis cuantitativo y research de sistemas de inversión, backtestings, screeners, bots de trading entre otros ejemplos.

Me enfoqué en la parte práctica ya que entiendo que la programación es una disciplina que se aprende y se asimila con horas y horas de sentarse a resolver problemas. Esto es 80% práctica y 20% teoría, aprender a programar es como aprender a manejar, la cantidad de horas al teclado es lo que hará la diferencia.

Creo que tanto la programación como las finanzas son dos campos importantes en la formación profesional y absolutamente imprescindibles para la planificación personal laboral y patrimonial en un mundo cada vez mas interesante, complejo y competitivo. Por eso espero que sea de utilidad y sea la puerta de entrada que termine acercando a mucha gente a este mundo tan fascinante y desafiante de la programación y de las finanzas cuantitativas.