

# **ANEXO I**

## **MODELO DE PORTADA**



<b>TRABAJO FIN DE GRADO</b>
<b>GRADO EN ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS</b>
<b>CURSO ACADÉMICO 2020/2021</b>
<b>CONVOCATORIA JUNIO</b>

<b>TÍTULO: APLICACIONES FINANCIERAS PROGRAMADAS EN PYTHON</b>
---

**APELLIDOS/NOMBRE ESTUDIANTE: GARRIDO RODRÍGUEZ, REBECA**

**DNI: 53460043Q**

**GRADO/DOBLE GRADO QUE CURSA: GRADO EN ADMINISTRACIÓN Y  
DIRECCIÓN EMPRESAS (SEMPRESENCIAL)**

**APELLIDOS/NOMBRE TUTOR:**

Aparicio, Adolfo
------------------

Fecha:27 de mayo de 2021

Revisión de las principales funciones financieras utilizadas en Excel y su programación en lenguaje Python. Análisis de las funciones de valor actual, valor final, pagos periódicos, tasa, número de periodos, TIR, VAN, entre otras, así como la creación de cuadros de amortización de préstamos. Extracción de información de la página de Yahoo Finance para el cálculo de las Betas de una compañía. Automatización de informes en Word.

## Contenido

1	INTRODUCCIÓN .....	4
2	PLANTEAMIENTO DEL TRABAJO .....	5
2.1	Objeto del Estudio .....	5
2.2	Metodología .....	5
3	DESARROLLO .....	8
3.1	Matemáticas Financieras y Finanzas.....	8
3.1.1	INTERES SIMPLE.....	9
3.1.2	INTERES COMPUESTO.....	12
3.1.3	TASA, NPER, VA Y VF.....	15
3.1.4	LAS RENTAS.....	17
3.1.5	DEPÓSITOS DE RENTA FIJA .....	19
3.1.6	ANÁLISIS DE PROYECTOS / INVERSIONES .....	24
3.1.7	LETRAS DE CAMBIO, LETRAS DEL TESORO, BONOS Y OBLIGACIONES .....	33
3.1.8	PRÉSTAMOS .....	37
3.1.9	EXTRACCIÓN DE DATOS WEB Y CÁLCULO DE BETAS .....	42
3.1.10	AUTOMATIZACIÓN DE INFORMES.....	44
4	CONCLUSIONES .....	46
5	ANEXOS.....	46
5.1.1	interés simple .....	47
5.1.2	interés compuesto .....	47
5.1.3	Uso de VF, VA, Tasa y Nper.....	48
5.1.4	Rentas .....	49
5.1.5	Depósitos .....	50
5.1.6	VAN & TIR .....	50
5.1.7	Letras de cambio, letras del tesoro y bonos.....	53
5.1.8	Préstamos .....	53
5.1.9	Beta de compañías .....	54
5.1.10	Automatización de informes .....	55
6	1. Características del Ibex35 .....	57
7	Referencias .....	58
8	Bibliografía.....	59

## 1 INTRODUCCIÓN

Podemos decir que actualmente las empresas se encuentran ante mercados globalizados donde obtener la más mínima ventaja frente a sus competidores puede suponer el éxito, o por el contrario el fracaso de un negocio. En muchos casos, estas empresas generan un gran volumen de datos, y convertir estos datos en información puede marcar una gran diferencia.

Esta información, en muchos casos se puede obtener en tiempo real, permite a las empresas saber cuál es el producto más vendido, el número de unidades vendidas, el volumen de ventas, la satisfacción de sus clientes o controlar los stocks y la logística de sus negocios. Por tanto, la digitalización es un pilar fundamental en el que los negocios deben apoyarse para maximizar sus beneficios.

Por otro lado, la información financiera nos permite elaborar informes acerca de la salud de una empresa, que cuenta con determinadas herramientas como la cuenta de Pérdidas y Ganancias, el Balance de Situación o los ratios financieros, que hacen la función de termómetros que te indican que algo puede no estar funcionando en el negocio.

Las matemáticas financieras nos permiten realizar un análisis cuantitativo acerca de la viabilidad económica de un proyecto de inversión o una financiación, obteniendo la información del riesgo que supone y por lo tanto supone una ayuda a la hora de tomar la decisión más correcta. Igualmente, nos permite optimizar el presupuesto mediante el análisis de los ingresos y costes, realizar proyecciones, elaboración de cuadros de amortización de préstamos que nos ayudan a planificar el ahorro, entre otros

Las empresas cuentan con software que les ayudan en este tipo de tareas, en función de su tamaño y su capacidad de inversión, sus programas podrán abarcar distintos aspectos del negocio, estar integrados y serán capaces de extraer datos e informes más o menos detallados y relacionados. Actualmente existen numerosos programas, como SAP y herramientas como Power BI o Google Analytics orientadas a este objetivo. Pero si bien hay una herramienta presente, independientemente del tamaño, el tipo de negocio, la capacidad de inversión o cualquier otra característica es EXCEL.

Excel se ha hecho un hueco en todas las empresas debido a su versatilidad, y fácil manejo, aunque si queremos sacarle el máximo partido hay que tener conocimientos más profundos de la herramienta. Cuenta además con un módulo financiero que desarrolla las fórmulas de las matemáticas financieras más comunes, permitiendo realizar cálculos como cuadros de amortización de préstamos, cálculos de intereses o la rentabilidad de un proyecto de forma sencilla.

Sin embargo, en muchas ocasiones nos podemos encontrar con tareas repetitivas que consumen tiempo efectivo de trabajo, por lo que tenemos que recurrir a macros o programaciones en visual basic, y una vez desarrolladas, es posible que otra persona

necesite manejar el fichero, y se produzcan cambios indeseados. Además, en algunas ocasiones, las fórmulas y la lectura de datos procedentes de otros sistemas (desde una web, ERP de la empresa o similar) ralentiza los cálculos, bloqueando el sistema en y reduciendo nuestra productividad.

Existen en el mercado numerosas herramientas, totalmente gratuitas, de código abierto con comunidades de usuarios en todo el mundo investigando, colaborando y aportando nuevas utilidades y funciones, de acceso libre, totalmente compatibles con Excel, que puede permitirnos aumentar nuestro rendimiento, realizar cálculos de una manera más sencilla, rápida y eficaz o automatizar tareas, generar informes, realizar análisis u cualquier otra tarea sin necesidad de renunciar a ninguna de ellas, sino complementarlas.

Para el desarrollo de este trabajo, he elegido una herramienta gratuita Ananconda, y la aplicación Jupyter Notebook, donde hemos pretendido desarrollar las fórmulas más comunes de Excel en lenguaje Python, y darles aplicación en el ámbito financiero con distintos ejemplos, para finalizar con la elaboración de un informe automatizado en Word, para mostrar así una herramienta alternativa y sus posibles aplicaciones en el mundo real.

## 2 PLANTEAMIENTO DEL TRABAJO

### 2.1 Objeto del Estudio

El objeto de estudio de este trabajo es realizar un repaso por de las fórmulas matemáticas financieras más comunes, usadas en el ámbito financiero para el análisis de viabilidad de proyectos, cálculos de intereses o amortizaciones de préstamos, realizando su desarrollo en lenguaje Python a través de la herramienta Jupyter Notebook, buscando ejemplos de aplicación.

Este trabajo no pretende profundizar en el ámbito más técnico de las herramientas ni del lenguaje, sino que pretende ser una muestra de la empleabilidad de los lenguajes de programación en el ámbito financiero y buscar las ventajas que nos pueda ofrecer para optimizar nuestros recursos.

Este trabajo tampoco pretende ser un curso o tutorial en el uso de este tipo de herramientas o desarrollo de lenguaje Python, para ello hay multitud de tutoriales en diferentes plataformas como YouTube o Udemy, totalmente gratuitos y de libre acceso para llegar hasta los conocimientos que deseemos adquirir.

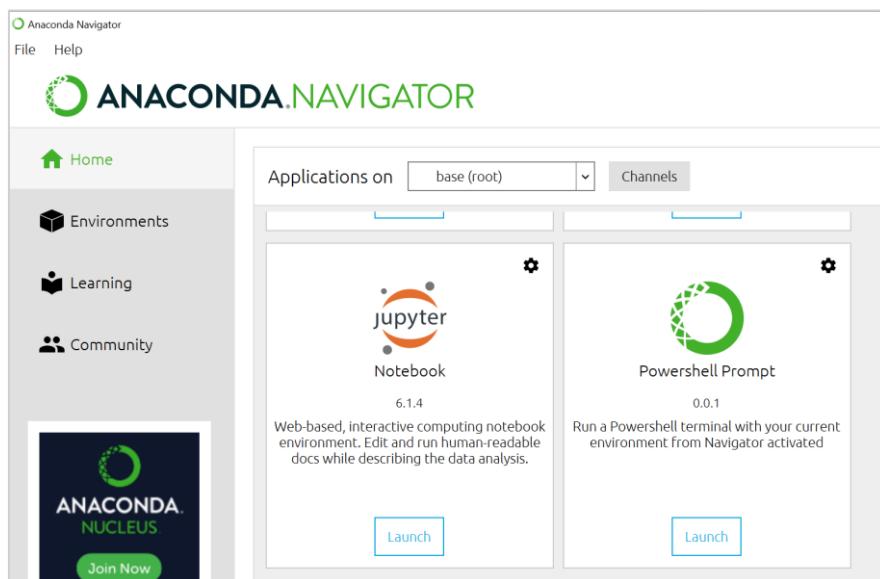
### 2.2 Metodología

Para el desarrollo de este trabajo se han utilizado las siguientes herramientas, todas ellas cuentan con una versión libre gratuita, disponibles en los enlaces indicados:

- Anaconda. – Es una Suite de código abierto que te permite la instalación de librerías de forma sencilla, y dispone de un navegador con diferentes herramientas para la elaboración de código, ya sea en Python o R.

En el siguiente enlace podemos descargarlo de forma gratuita:  
<https://www.anaconda.com/products/individual>

Su aspecto es el siguiente:



El navegador dispone además de documentación y cursos que nos facilitan amplia información de diferentes aspectos relacionados tanto con el uso de la herramienta, como de librerías o paquetes.

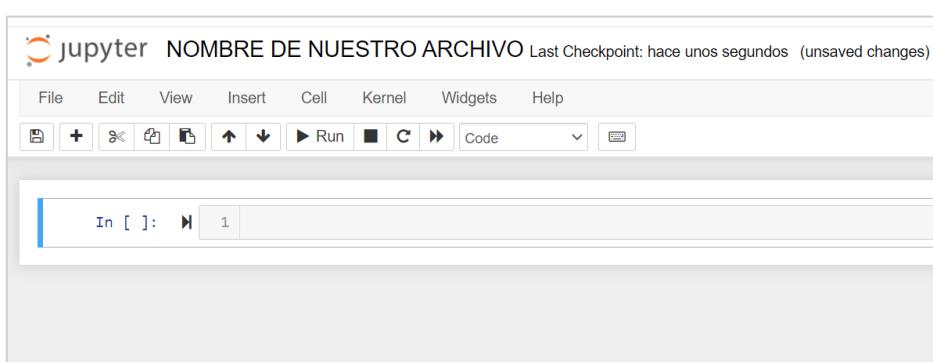
- **Jupyter Notebook.** – Python es un lenguaje de programación, y por tanto podemos utilizar diferentes herramientas para escribir código. Entre las más conocidas está Sublime Text, Spyder, o Atom que presentan un aspecto típico de pantalla negra con código, habitual en programadores informáticos. Sin embargo, en nuestro caso, dado que este trabajo no tiene un enfoque técnico hemos buscado una herramienta más “amigable” para usuarios poco avanzados, que nos ofrece exactamente el mismo rendimiento y dispone de herramientas internas que pueden servir de ayuda en determinados momentos.

Esta herramienta es accesible desde Anaconda Navigator, está incluida de forma totalmente gratuita, y se desarrolla en el explorador que tengamos disponible, Chrome o Edge.

También se puede descargar directamente en el ordenador si no queremos descargar Anaconda, en el siguiente enlace: <https://jupyter.org/install>

En este enlace podemos encontrar distintos interfaces a parte de Jupyter Notebook, como son Jupyter Lab o Jupyter Console.

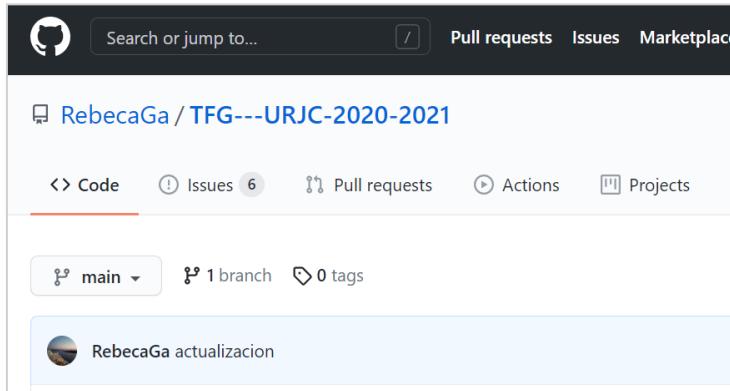
Esta es la imagen que presenta la herramienta una vez abierta:



- GitHub. – para compartir la información con mi tutor, y todo aquel que lo necesite, he creado un repositorio en la página GitHub que nos ha permitido un buen flujo de trabajo a la hora de compartir los avances.

Este es el enlace al repositorio: <https://github.com/RebecaGa/TFG---URJC-2020-2021.git>

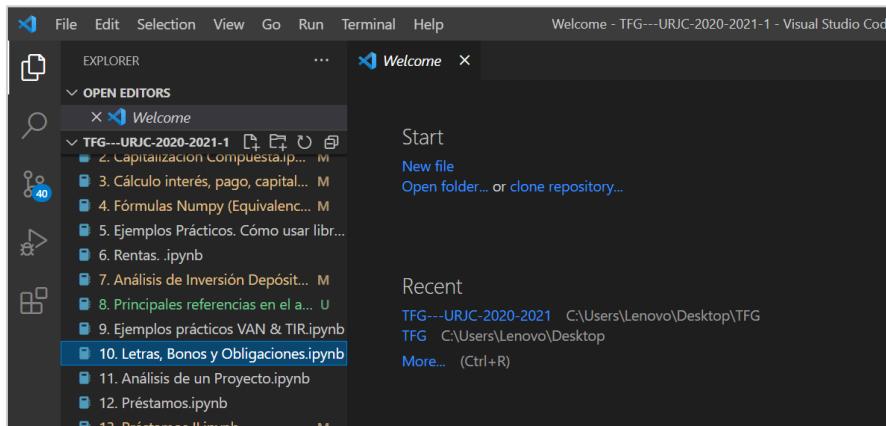
Y la imagen:



- Visual Studio Code. – Por último, para automatizar la actualización del código en GitHub se utilizó Visual Studio Code que me permitió ir actualizando cada avance del proyecto de una forma dinámica y sencilla.

Esta herramienta también es gratuita, y se puede descargar en el siguiente enlace: <https://code.visualstudio.com/download>

Su apariencia es:



Con todas estas herramientas, hemos podido ir trabajando poco a poco sobre las distintas fórmulas matemáticas financieras, desarrollando código que permitiese realizar el cálculo de forma automática sin la necesidad de conocer previamente dichas fórmulas.

En cuanto al método empleado, se comenzó por el estudio de las fórmulas más comunes como la del interés simple, para ir desarrollando cálculos más complejos hasta llegar al cálculo de la Beta de una compañía, que requiere de conocimientos estadísticos. Así mismo, se han estudiado las fórmulas contenidas en el apartado financiero de Excel, dado que el principal objetivo de este trabajo es comparar ambas herramientas en la elaboración de estudios de viabilidad de proyectos, cálculos de cuadros de amortizaciones de préstamos o cualquier otra tarea relacionada.

## 3 DESARROLLO

### 3.1 Matemáticas Financieras y Finanzas

Para entender las matemáticas financieras es necesario entender el valor del dinero en el tiempo. A medida que transcurre el tiempo el dinero pierde valor, debido a la inflación y, en consecuencia, existe una pérdida de poder adquisitivo.

Nacen paralelas al comercio, ante la necesidad de realizar los cálculos necesarios para el estudio del valor del dinero en el tiempo derivados los intercambios de bienes y servicios. Mediante la combinación de tres factores, como capital, tasa y tiempo somos capaces de calcular los rendimientos o intereses, además de establecer métodos que nos permiten la evaluación un proyecto o inversión, haciendo posible la toma de decisiones.

Podemos decir, que las empresas familias o estado, a través de las Finanzas pueden realizar, en situaciones de incertidumbre, un estudio para tomar la mejor decisión de inversión, ahorro o gasto, a través de la elección de distintos recursos como bonos, acciones, adquisiciones de bienes de capital, edificio o inversión en infraestructuras.

Por tanto, las finanzas a través de las matemáticas nos permiten estimar la rentabilidad o coste de un proyecto o inversión dotándonos de la información suficiente para tomar una decisión. Nos permiten realizar controles sobre el gasto optimizando el presupuesto, realizar proyecciones a futuro para tener una visión a largo plazo, elaborar cuadros de amortización que sin duda nos ayudan a planificar el ahorro, y analizar la inflación, permitiendo conocer el valor real del dinero en diferentes momentos en el tiempo, siendo este su objetivo principal.

Para elaborar estos estudios requerimos de herramientas, siendo Excel un programa informático que permite trabajar con datos numéricos, realizar cálculos aritméticos, aplicar fórmulas de mayor complejidad e incluso realizar operaciones estadísticas, facilitando el análisis de los datos.

Python por su parte es un lenguaje de programación orientado a objetos que se caracteriza por su elegante sintaxis, su código legible y su facilidad. Es un lenguaje interpretado, dinámico, multiplataforma y de código abierto. Si hay algo que diferencia a Python de otros lenguajes de programación es el no uso de símbolos para los operadores lógicos, escritos como not, or y and. Sus variables se definen de forma dinámica, es decir, sin necesidad de especificar el tipo de antemano y puede tomar distintos valores, asignados mediante un símbolo =.

A continuación, haremos un repaso de las distintas fórmulas que podemos usar, su base matemática, la forma de emplearlas en Excel y distintas formas de emplearlas en Python.

### 3.1.1 INTERES SIMPLE

Hemos visto que las matemáticas financieras están fundamentadas en 3 factores, a saber, el capital, la tasa o interés y el tiempo. La fórmula del interés simple nos permite calcular el rendimiento de un capital, al que se le aplica una tasa de forma constante en el tiempo, sin que se le añadan nuevos periodos.

Este interés puede ser pagado o abonado, y dado que no se incorporan al capital, se devengan al final del periodo.

Su fórmula es la siguiente:

$$C_n = C_0 * (1 + i * n)$$

Donde:

$C_n$  es el capital final

$C_0$  es el capital inicial

i es el tipo de interés

n corresponde a los periodos

Lo más importante a tener en cuenta en este cálculo es cerciorarnos que todas las unidades se encuentran en la misma medida, es decir, si el tiempo está expresado en años, debemos asegurarnos de que el tipo de interés está expresado en la misma forma, por tanto, deben ser homogéneos. Si no lo son, debemos transformar bien el tiempo o bien el tanto por ciento de interés, siendo indiferente para la obtención del resultado que factor transformemos.

Entre las características más importantes podemos indicar que el capital inicial se mantiene constante, siendo la misma cantidad a lo largo de todos los periodos de la operación, aplicando el tipo de interés únicamente al capital invertido. Los intereses que se van generando no se acumulan.

No es habitual encontrar este tipo de interés en la vida diaria, pero usaremos este anuncio de un depósito a plazo fijo de una famosa entidad bancaria para realizar un ejemplo.

**Obtén una mayor rentabilidad con tu depósito.**

Si combinas un depósito a plazo fijo con un fondo de inversión te ofrecemos una mejor rentabilidad.

- ✓ Hasta un 0,65 % TAE (0,65 % TIN).
- ✓ Desde 600 €.
- ✓ 13 meses de duración.

[Contratar](#)

El anuncio nos indica que si mantenemos al menos 600 euros durante un periodo de 13 meses en el banco, este nos remunerará a un tipo de interés del 0,65 %. Vamos a comprobar cuales serían los intereses que obtendríamos:

En primer lugar, tendríamos que convertir nuestro interés expresado en años, en interés expresado en meses:

$$i_m = \frac{i}{m} \quad i_{12} = \frac{0,0065}{12} \quad i_{12} = 0,00054167$$

Posteriormente podemos usar la fórmula de interés simple para calcular cuánto con cuánto dinero nos va a remunerar BBVA el mantener nuestros ahorros en su entidad:

$$\text{Intereses} = 600 * (0,00054167) * 13 = 4,225026 \text{ euros}$$

Dentro del interés simple podemos encontrarnos con tipos de interés equivalentes, que son aquellos que, aplicados a la misma cuantía, es decir al capital inicial, durante el mismo periodo de tiempo nos generan los mismos intereses. Estos se relacionan entre sí de forma proporcional.

Para conocer el tanto equivalente a un periodo en el interés simple podemos utilizar la siguiente fórmula:

$$i_m = \frac{i}{m}$$

A partir de las fórmulas que hemos presentado, no solo podemos conocer los intereses o el valor final, también podemos despejar cualquiera de sus variables teniendo todos los datos, así disponiendo del capital inicial, el capital final y el periodo de tiempo podemos obtener el tipo de interés que se le aplica a la operación, o si disponemos del capital inicial, el capital final y el interés, podemos obtener el tiempo que ha transcurrido.

Veamos ahora un ejemplo aplicado a Excel: Calcular los intereses anuales obtenidos a un 10% de interés sobre un capital inicial de 10.000 euros durante 4 años mediante interés simple.

C0	10.000,00 €	AÑO	INTERESES
i	10%	1	1.000,00 €
n	4	2	1.000,00 €
Cn		3	1.000,00 €
Cn		4	1.000,00 €
		Total	4.000,00 €

Siendo la fórmula empleada  $C0*(1+i*n)$  para calcular el capital final, y para calcular los intereses anuales  $C0*i$ , sumando posteriormente los cuatro años para obtener los intereses totales.

Vamos a ver la misma operación realizada en Python:

```
In [2]: 1 C0 = 10000
         2 i = 0.10
         3 n = 4
         4
         5 Cn = C0 * (1 + i * n)
         6 Cn
Out[2]: 14000.0
```

Pero vayamos un paso más allá. ¿Por qué no crear una función que realice las operaciones necesarias para calcular el valor final de una operación empleando la fórmula de la capitalización simple?

Para su elaboración, se plantean las distintas posibilidades que pueden existir a la hora de tener que realizar el cálculo. Sabemos, que para el uso de la fórmula de interés simple los factores tiempo y tipo de interés deben ser homogéneos, por lo que en primer lugar debemos solventar las posibles diferencias que puedan surgir en este sentido.

El objetivo es determinar la frecuencia de los períodos, así como los períodos en sí, por ello se solicita al usuario dos entradas relacionadas. En la primera, solicitamos el tipo de periodo, que establece la posibilidad de obtener los rendimientos de forma mensual, trimestral, semestral o anual.

```
tipo = input(''Indique que tipo de periodo:
    m = mensual
    a = anual
    t = trimestral
    s = semestral '').lower()
```

Después, solicitamos el tipo de interés de la operación expresado en tanto Nominal anual, para poder calcular el equivalente en función de la frecuencia.

Sabremos que, si hemos expresado una frecuencia mensual,  $i_m$  será  $i_{12}$ ; si la frecuencia es trimestral  $i_m$  será  $i_4$ ; para una frecuencia semestral estaremos ante  $i_2$ , de manera que podemos reinterpretar la fórmula de interés simple:

$$C_n = C_0 * \left(1 + periodos * \frac{i}{frecuencia}\right)$$

De esta manera obtenemos una fórmula genérica, que nos permite calcular cualquier capital final. El código de la operación quedaría de la siguiente forma:

```
def CalculaFinanciera():
    mes = ['m', 'mes', 'mensual', 'uno']
    año = ['a', 'anual', 'año']
    trimestral = ['t', 'trimestral', 'trimestre', 'tres']
    semestral = ['s', 'semestre', 'semestral', 'seis']
    try:
        tipo = input(''Indique que tipo de periodo:
            m = mensual
            a = anual
            t = trimestral
            s = semestral '').lower()
        capitalInicial = float(input('Importe a invertir: '))
        tantoNominal = float(input('Introduzca el tipo de interés Nominal anual (Ejemplo 8%: 0.08) : '))
        if tipo in mes:
            periodoTemporal = 'Meses'
            frecuencia = 12
        elif tipo in año:
            periodoTemporal = 'Años'
            frecuencia = 1
        elif tipo in trimestral:
            periodoTemporal = 'Trimestres'
            frecuencia = 4
        elif tipo in semestral:
            periodoTemporal = 'Semestres'
            frecuencia = 2
        periodos = float(input('Número de períodos (en {}): '.format(periodoTemporal)))
        vf = capitalInicial * (1 + periodos * (tantoNominal / frecuencia))
        print("Su inversión tendrá un valor futuro de: {}".format(vf))
    except ValueError:
        print('Error!, solo se admiten numeros')
```

Esta versión la podemos simplificar, mediante la realización de un aviso, tal como se ha diseñado en este otro código:

```

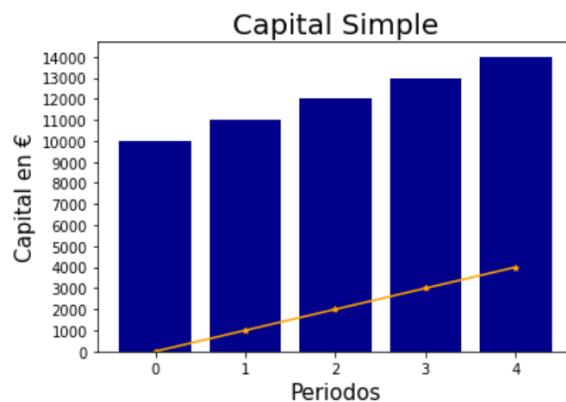
1 F calculo_simple(C0, i, n):
2     Cn=C0*(1+i*n)
3     return round(Cn,2)
4
5 int("""          El tipo de interés (i) y el número de periodos (n) que introduzca deben tener :
6             Si el tipo de interés es anual, entonces el número de periodos han de ser años.
7             Si se trabaja con meses, entonces el tipo de interés introducido ha de ser un tipo mensual.
8
9             Por ejemplo, si el tipo de interés es del 12% anual y la operación de inversión a interés
10            dura 18 meses, indicaremos que el tipo de interés es i=0,12 y el número de años es n=1,5
11            """
12
13     = float(input('Importe a invertir: '))
14     = float(input('Introduzca el tipo de interés (Ejemplo 8%: 0.08) : '))
15     = float(input('Número de periodos: '))
16
17 int('')                                El capital final obtenido es', calculo_simple(C0, i, n))
18 int('')

```

Python también nos permite realizar gráficas, para ello cuenta con una serie de librerías con código ya prediseñado y funciones guardadas, para que todo usuario pueda utilizarlo de forma totalmente gratuita. Estas librerías en ocasiones hay que instalarlas, pero la mayor parte de las veces tan solo hay que activarlas mediante la llamada “import”, para posteriormente utilizarlas como deseemos.

La librería para gráficos más usada es matplotlib, donde se puede encontrar diversa documentación acerca de las posibilidades que ofrece en el siguiente enlace:  
<https://matplotlib.org/>

Si representamos gráficamente nuestro ejemplo, veríamos que nuestros intereses aumentan la misma cantidad año tras año, para llegar al montante final:



### 3.1.2 INTERES COMPUESTO

Cuando hablamos de capitalización compuesta en una operación financiera, los intereses se van a ir acumulando al capital para los siguientes períodos, generando a su vez nuevos intereses. El capital final estará formado por tanto del capital inicial más los intereses que se van generando periódicamente, de los cuales podremos disponer al final del periodo.

Este tipo de operaciones se caracterizan por acumular los intereses al capital inicial a medida que se van abonando, y estos a su vez producen nuevos intereses en los siguientes períodos.

Su fórmula es la siguiente:

$$C_n = C_0 * (1 + i)^n$$

Donde:

$C_n$  es el capital final

$C_0$  es el capital inicial

$i$  es el tipo de interés

$n$  corresponde a los períodos

Este tipo de interés es el más común, empleado en la mayoría de las operaciones financieras. Su efecto es exponencial, lo que conlleva un aumento de los beneficios o del montante de la deuda, puesto que como hemos visto los propios intereses generan a su vez nuevos intereses.

Hay que tener en cuenta la homogeneidad igualmente entre el interés y los períodos, pero debemos prestar especial cuidado a la hora de calcular los tantos por cientos equivalentes, dado que estos en interés compuesto se relacionan de forma exponencial frente a la relación lineal que mantienen en el interés simple.

Por ello, debemos tener presente la fórmula que calcula intereses compuestos equivalentes que es:

$$i_n = (1 + i)^{\frac{1}{n}} - 1$$

Es por ello por lo que debemos prestar especial atención a las informaciones bancarias, y los datos que aparecen en la letra pequeña para determinar cual es el tipo de interés correcto que debemos emplear.

Vamos a ver un ejemplo gráfico. Observemos este anuncio de Bankinter:



Como podemos observar el anuncio nos indica un 5% TAE, y justo al lado el número nos indica que existe una letra pequeña, que podemos ver a continuación:

<sup>1</sup> **Cuenta Nómica 5%:** Promoción válida hasta el 30/06/2021 o hasta un máximo de 40.000 nuevas cuentas actualmente domiciliada en Bankinter, que sean personas físicas, residentes en España, que no se hayan ni hayan sido titulares de una Cuenta Nómica/Pensión/Profesional en Bankinter durante los 12 meses anteriores al inicio de la promoción. Titular de una cuenta remunerada y como máximo ser titular de dos cuentas corrientes bonificadas. A esta cuenta corriente, que tenga condiciones económicas especiales tales como tarjetas de crédito sin comisión de encargo ni remunerar 5.000€. Primer año: tipo de interés nominal anual 4,94% (5%TAE). Segundo año: tipo de interés nominal anual 4,94% (5%TAE). Saldo en cuenta nómica diario de 5.000€, calculado para un periodo de liquidación de 180 días, remuneración 49,07€, 4º semestre 49,07€. Para acceder y conservar las ventajas de nuestra cuenta nómica, además de cumplir con los requisitos establecidos en la promoción, deberá mantener un saldo en la cuenta nómica superior a 5.000€ al menos tres recibos domiciliados en el trimestre, así como que la tarjeta asociada a la cuenta nómica esté en uso. El plazo máximo para realizar el primer ingreso, que deberá ser igual o superior al declarado en la solicitud, no podrá超过 3 meses desde la contratación de la Cuenta Nómica.

Según nos indica, el 5% TAE equivale a un interés nominal anual de 4,94%, por tanto, este es el tipo de interés que debemos usar en caso de querer realizar cualquier cálculo en referencia a este producto.

Una vez aclarado los tantos equivalentes, vamos a ver un ejemplo práctico de capital compuesto: Calcular los intereses anuales obtenidos a un 10% de interés sobre un capital inicial de 10.000 euros durante 4 años mediante interés compuesto. Como podemos observar, el enunciado es el mismo, y vamos por tanto a comparar los resultados.

Si realizamos los cálculos mediante Excel, podemos observar que obtenemos 641 €, más cantidad que mediante interés simple.

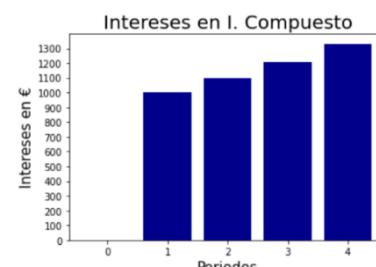
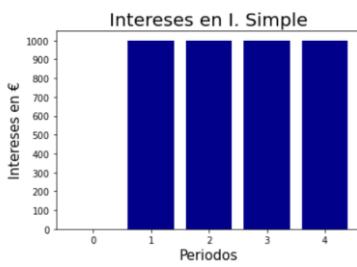
<b>C<sub>0</sub></b>	10.000,00 €	<b>AÑO</b>	<b>INTERESES</b>
<b>i</b>	10%	1	1.000,00 €
<b>n</b>	4	2	1.100,00 €
<b>C<sub>n</sub></b>	14.641,00 €	3	1.210,00 €
		4	1.331,00 €
		<b>Total</b>	<b>4.641,00 €</b>

Si elaboramos la fórmula en Python, vemos:

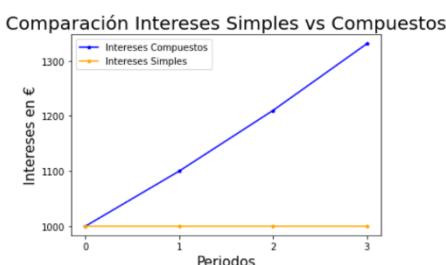
```
In [2]: 1 C0 = 10000
         2 i = 0.10
         3 n = 4
         4
         5 Cn = C0 * (1+i)**n
         6 print(round(Cn,2))
```

14641.0

Comprobando que obtenemos el mismo resultado en ambos formatos. Pero veamos las siguientes gráficas:



En ellas podemos apreciar como en los intereses generados mediante la fórmula de capital simple son lineales, cada año obtenemos el mismo rendimiento, frente a los intereses obtenidos mediante la fórmula de capital compuesto que van aumentando año tras año.



En esta imagen, podemos apreciar de igual forma como los intereses compuestos crecen de forma exponencial, frente al crecimiento lineal de los intereses en capitalización simple.

### 3.1.3 TASA, NPER, VA Y VF

El hecho de emplear la palabra simple en una operación financiera no implica necesariamente que se utilice interés simple, en la mayor parte de las ocasiones de hecho se utilizará el interés compuesto. La operación simple hacer referencia a la carencia de capitales intermedios, existe un capital inicial y un capital final, pero no se incorporan nuevos capitales a lo largo de la operación.

Excel dispone de una serie de fórmulas que permiten despejar cualquiera de las variables existentes en interés compuesto, así para el capital final, disponemos de la función VF (en inglés la encontraremos como FV), que nos devuelve el valor futuro de una inversión, a una tasa de interés determinada, siendo constante en un determinado periodo de tiempo.

**VF(tasa; nper; pago; [va]; [tipo])** equivale a  $C_n = C_0 * (1 + i)^n$

Encontramos también la fórmula TASA (en inglés RATE), la cual nos devuelve la tasa de interés a partir del periodo y la anualidad. 1

**TASA(nper; pago; va; [vf]; [tipo]; [estimar])** que es equivalente a  $i = \left(\frac{C_n}{C_0}\right)^{\frac{1}{n}} - 1$

Otra de las funciones que podemos encontrar es NPER (con el mismo nombre en inglés) que nos devuelve el numero de periodos de una inversión, la duración de la operación financiera, en base a los pagos periódicos constantes y mediante una tasa de interés también constante.

**NPER(tasa; pago; va; [vf]; [tipo])** que es equivalente a  $n = \frac{\ln(\frac{C_n}{C_0})}{\ln(1+i)}$

Por último, encontramos la función VA (en inglés PV), que nos devuelve el valor actual de una inversión, que corresponde al valor total de los pagos futuros.

**VA(tasa; nper; pago; vf; [tipo])** que es equivalente a  $C_0 = C_n * (1 + i)^{-n}$

Estos argumentos significan:

Tasa (rate). - es el tipo de interés aplicado por periodo. Es importante que sea homogéneo con el tiempo.

Nper (nper). – es el número de periodos que va a durar la operación, puede tratarse de meses, años, trimestres, etc. Igualmente debe mantener una homogeneidad con el tiempo y la tasa.

Pago (pmt). – hace referencia a los pagos periódicos en caso de renta, si trabajamos con operaciones simple será 0.

VA (pv). – equivalente a C0 o capital inicial. Si aparece entre corchetes se deberá indicar en negativo

Tipo (when). – Es totalmente opcional, en caso de omisión se considera 0, que implica que el devengo de intereses es al final del periodo. Si indicamos 1 estaremos indicando que los intereses se generan al principio del periodo.

VF (fv).— Equivalente a Cn o valor final. Si aparece entre corchetes deberá indicarse en negativo.

Estimar. – De aplicación únicamente en la fórmula tasa, sirve para indicar el entorno en que se encuentra la tasa. En caso de omisión se entiende 10 por ciento.

Como hemos visto Python dispone de librerías, que son conjuntos de módulos que contienen las operaciones más comunes en programación de uso diario, que están implementadas. Pero en ocasiones, la comunidad de usuarios desarrolla nuevas librerías que ponen a disposición del resto de personas de forma libre y gratuita principalmente en PyPI (Python Package Index). Es el caso de la librería Numpy Financial, cuya documentación podemos acceder en el siguiente enlace: <https://numpy.org/numpy-financial/latest/>

Esta librería contiene una serie de funciones con la misma sintaxis que las fórmulas que hemos visto anteriormente en Excel, de modo que su uso se hace de forma bastante intuitiva. Podemos ver las funciones en el siguiente cuadro:

## Functions

<code>fv(rate, nper, pmt, pv[, when])</code>	Compute the future value.
<code>ipmt(rate, per, nper, pv[, fv, when])</code>	Compute the interest portion of a payment.
<code>irr(values)</code>	Return the Internal Rate of Return (IRR).
<code>mIRR(values, finance_rate, reinvest_rate)</code>	Modified internal rate of return.
<code>nper(rate, pmt, pv[, fv, when])</code>	Compute the number of periodic payments.
<code>npv(rate, values)</code>	Returns the NPV (Net Present Value) of a cash flow series.
<code>pmt(rate, nper, pv[, fv, when])</code>	Compute the payment against loan principal plus interest.
<code>ppmt(rate, per, nper, pv[, fv, when])</code>	Compute the payment against loan principal.
<code>pv(rate, nper, pmt[, fv, when])</code>	Compute the present value.
<code>rate(nper, pmt, pv, fv[, when, guess, tol, ...])</code>	Compute the rate of interest per period.

En este caso requiere instalación previa, para después ejecutar la llamada de la librería antes de usarla. Para su instalación escribiremos el siguiente código, para a continuación realizar la llamada.

```
In [1]: ┌ 1 pip install numpy-financial
      ┌ 2
```

```
In [2]: 1 import numpy_financial as npf
```

Un ejemplo de su utilización sería:

```
In [15]: 1 i = float(input("Interes: "))
2 n = float(input("Plazo: "))
3 C0 = float(input("Capital inicial: "))
4
5 fv = npf.fv(rate=i, nper=n, pmt= 0,pv=-C0, when='end')
6 print("El capital final en esta operación es {}".format(round(fv,2)))

Interes: 0.10
Plazo: 4
Capital inicial: 10000
El capital final en esta operación es 14641.0
```

\*Nótese que, para llamar a la función, previamente debemos escribir la letra *npf*, dado que le hemos asignado dichas letras al llamar a la función.

### 3.1.4 LAS RENTAS

Una renta financiera hace referencia a un conjunto de capitales, cuyos vencimientos se producen en un intervalo de tiempo, cuyos elementos fundamentales son por un lado el origen, al otro extremo el final, el tiempo transcurrido que hace referencia a la duración, los términos que son cada uno de los capitales financieros que componen una renta, el periodo que indica el tiempo entre dos capitales consecutivos y que suele ser constante, y finalmente el valor financiero o el valor capital de una renta. Por tanto, la renta será un ingreso constante, que tenga un origen y un final (aunque existen rentas perpetuas) que se producirá por periodos.

Existen varias clasificaciones de las rentas como que planteamos a continuación (Aparicio, [www.masterfinanciero.es](http://www.masterfinanciero.es) s.f.):

- Primera Clasificación:
  - Rentas discretas. - su cuantía vence en un momento concreto
  - Rentas continuas. - suele hacer referencia a rentas teóricas que se suelen encontrar en manuales de Finanzas, y trabajan con una distribución de cuantía.
- Segunda Clasificación:
  - Rentas unitarias. - su cuantía es 1 euro
  - Rentas de cuantía variable. - en cuyo caso la renta no es constante. Dentro de estas podemos encontrar:
    - Rentas de progresión aritmética
    - Rentas de progresión geométrica.
- Tercera Clasificación
  - Rentas inmediatas. – cuya valoración se realiza al principio y al final de la renta.
  - Rentas diferidas. – cuya valoración se realiza en un punto anterior al origen de la renta.
  - Rentas anticipadas. – cuando el punto de valoración se encuentra posterior al origen de la renta.
- Cuarta Clasificación
  - Rentas temporales. – cuyo número de términos es finito
  - Rentas perpetuas. - cuyos términos tienden a infinito
- Quinta Clasificación
  - Rentas prepagables. – en las que la cuantía se abona al inicio del periodo
  - Rentas pospagables. – cuya cuantía vence al final del periodo. (Aparicio, [www.masterfinanciero.es](http://www.masterfinanciero.es) s.f.)

Lo que más nos interesa saber en el caso de las rentas es su valor financiero en un determinado momento del tiempo. Por ejemplo, cuánto dinero necesitamos ahorrar si queremos adquirir un coche y pagarlo al contado, o a la inversa, que dinero necesito tener

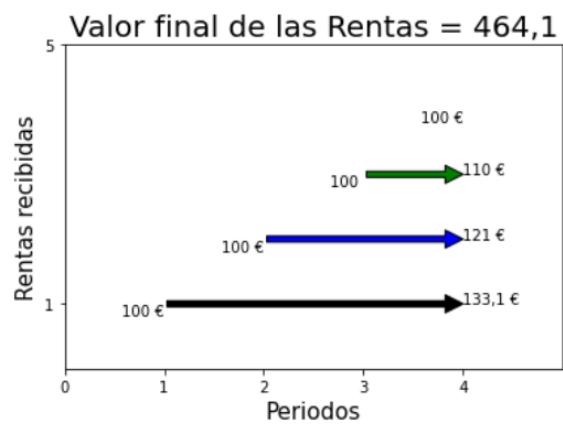
hoy para poder recibir una renta mensual de 100 €. En definitiva, lo que hacemos es trasladar financieramente todas las cuantías a un punto determinado del tiempo, si son anteriores se capitalizan y si son posteriores se descuentan. Su valoración se suele realizar utilizando la ley de capitalización compuesta.

En el caso de tener que capitalizar utilizaremos la fórmula del capital final (VF o FV), mientras que si tenemos que descontar emplearemos la fórmula del capital inicial (VA o PV).

Veamos un ejemplo. Calcular el valor final de una renta pospagable de 4 términos de una cuantía constante de 100 €, valorada al 10% (Aparicio, www.masterfinanciero.es s.f.):

```
In [8]: 1 n = 4
2 pmt = 100
3 tasa = 0.10
4 #m = 12
5 #im = (1+tasa)**(1/m)-1
6
7
8 #vamos a crear una lista
9 periodos = range(n)
10 renta = [0]
11
12 for i in periodos:
13     a = pmt*(1+tasa)**i
14     print(round(a,2))
15     renta.append(a)
16
17 renta_total = sum(renta)
18 print(round(renta_total,2))

100.0
110.0
121.0
133.1
464.1
```

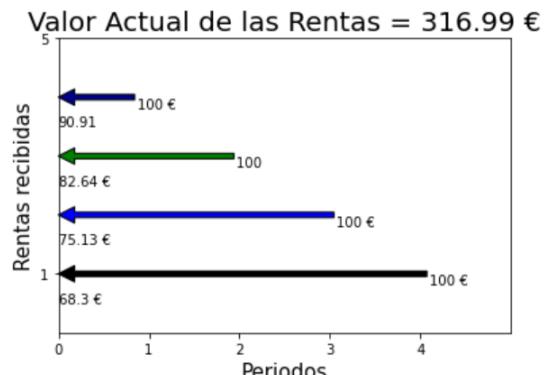


En este caso tenemos que capitalizar nuestras rentas, para ello tenemos que trasladar al año 4 las rentas que vamos obteniendo mes a mes. El año 0 no obtenemos rentas, el año 1 obtenemos 100 euros que si proyectamos al año 4, el valor final serán de 133,10 euros. El año 2 obtenemos 100 euros de renta, que si proyectamos al año 4, el valor final de nuestra renta será de 121. El año 3 obtendremos otros 100 euros, que proyectados obtendremos 110 euros, para finalmente en el año 4 obtener la última renta de 100 euros. Si sumamos las rentas y sus intereses, finalmente obtenemos 464,1 €

En caso de querer conocer el valor actual de las rentas, debemos actualizar el valor al momento 0.

```
In [9]: 1 n = 4
2 pmt = 100
3 tasa = 0.10
4 #m = 12
5 #im = (1+tasa)**(1/m)-1
6
7
8 #vamos a crear una lista
9 periodos = range(n+1)
10 renta = [0]
11
12 for i in periodos:
13     a = pmt*(1+tasa)**-i
14     print(round(a,2))
15     renta.append(a)
16
17 renta_total = sum(renta)-pmt
18 print(round(renta_total,2))

100.0
90.91
82.64
75.13
68.3
316.99
```



En cualquier caso, podemos usar la librería existente numpy financial que contiene las fórmulas financieras de valor actual y valor final, además de las versiones para obtener la tasa o los pagos, tal como podemos ver en el siguiente ejemplo:

```
In [192]: 1 valor_actual = npf.pv(0.10,4,-100,0,0)
2 print("Valor actual de la renta", round(valor_actual,2))
3
4 valor_final = npf.fv(0.10,4,-100,0,0)
5 print("Valor final de la renta", round(valor_final,2))

Valor actual de la renta 316.99
Valor final de la renta 464.1
```

### 3.1.5 DEPÓSITOS DE RENTA FIJA

Los depósitos de renta fija son productos financieros, que consisten en la entrega de una determinada cantidad, normalmente a una entidad bancaria, durante un tiempo determinado por lo que a cambio obtenemos una remuneración. Al finalizar el plazo, recuperamos el capital invertido además de los intereses que se hayan generado.

Podemos encontrar depósitos a largo plazo o a corto, pero en cualquier caso son productos carentes de riesgo, y no hay que confundir en ningún caso con fondos de renta fija, los cuales si tienen cierto riesgo. El cálculo de dichos intereses se realiza mediante capitalización compuesta.

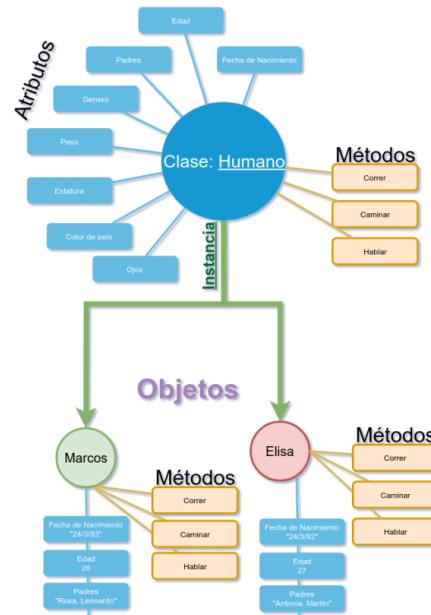
Aunque actualmente, debido al momento económico en el que nos encontramos, no hay una abundancia de este tipo de productos, hubo un tiempo en el que era habitual recibir publicidad de diversas entidades bancarias que ofrecían tipos de interés más que aceptables por periodos no muy elevados. Era tal la cantidad de oferta que era bastante costoso decidirse por uno u otro producto o entidad, así que disponer de una herramienta que nos permita un rápido análisis puede ser una ventaja a la hora de tomar una decisión.

Una de las ventajas que nos puede ofrecer Python es la rapidez a la hora de realizar cálculos, y nos permite construir funciones que nos permiten comparar diversos productos, en pocos segundos podremos conocer que opción se ajusta más a nuestras necesidades.

Python como hemos dicho anteriormente, es un programa orientada a objetos, y dispone de la opción de crear clases, que consiste en empaquetar datos y funciones de manera conjunta. Cada clase crea un nuevo tipo de objeto, al que se le pueden crear instancias. Cada una de estas instancias pueden tener a su vez atributos, y métodos que modifiquen su estado. Vamos a intentar a explicarlo de una forma más sencilla. Cuando creamos una clase estamos definiendo una plantilla a partir de la cual podemos crear diferentes objetos, llevándolo a la vida cotidiana, los vehículos serían nuestra clase en Python, y los coches, motos o camiones serían los objetos.

Los objetos en programación, son entidades que pueden tener un comportamiento, un estado, pueden contener información y pueden incluso realizar tareas. En nuestro ejemplo

anterior, podemos definir coche como un objeto, este puede tener comportamientos, como “arrancar” o “detenerse”, puede tener estados como “en marcha” o “parado”, puede contener información como “4 puertas” o “5 puertas” y finalmente puede realizar tareas tales como “comprobación del motor automática”. Los objetos tienen atributos, que podríamos definir como características. Y también tienen métodos, que podríamos definir como las acciones o tareas que el objeto puede realizar. Esto no implica que todos los objetos tengan que estar definidos en una clase. Con el siguiente gráfico creo que podemos comprender mucho mejor los conceptos explicados:



FUENTE: web <https://pythones.net/clases-y-metodos-python-oop/> (Laca s.f.)

Aprovechando esta característica que nos ofrece Python, nosotros vamos a crear una clase denominada “Depósitos”, que nos permitirá realizar una serie de análisis y comparaciones entre distintos depósitos para evaluar el que mejor se ajusta a nuestras necesidades.

Comenzamos definiendo una serie de funciones que nos ayudarán posteriormente a crear nuestros métodos, es decir las acciones que podrán realizar nuestros depósitos, de una forma más sencilla.

```

In [5]: 1 #Damos forma a determinadas funciones que después vamos a incorporar en nuestra class
2 #Empezamos por Los DEPÓSITOS
3
4 def VF(rate, nper, pmt, when=0):      #para conocer el importe que voy a obtener al final de periodo
5     VF = npf.fv(rate, nper, pmt, pv, when)
6     return VF
7
8 def RATE(nper, pmt, pv, fv, when=0):      #En caso de conocer el resto de los datos pero no el interés.
9     RATE = npf.rate(nper, pmt, pv, fv, when)
10    return RATE
11
12 def INTERESES(rate, nper, pmt, pv, when=0):      #Para conocer los intereses que genera mi depósito
13     pv = pv
14     VF = npf.fv(rate, nper, pmt, -pv, when)
15     INTERESES = VF - pv
16     return INTERESES
17
18 def TIEMPO(rate, pmt, pv, fv=0, when=0):      #Si quiero conocer el tiempo que debo mantener una inversión
19     TIEMPO = npf.nper(rate, pmt, -pv, fv=fv, when=when)
20     return TIEMPO
  
```

Aquí podemos comprobar que hemos definido una función para conocer el valor final de mi inversión, que estará compuesta por el capital que se ha invertido más lo intereses que se hayan generado. La hemos denominado “VF”, y requiere de atributos como rate, nper, pmt, pv o when (tipo de interés o tasa, número de periodos, importe de los pagos por periodos, capital inicial y momento de devengo de intereses, bien al inicio del periodo o bien al final).

La segunda función que hemos creado es para determinar el tipo de interés aplicado al depósito. La hemos denominado “RATE”, y requiere los atributos nper, pmt, pv, fv y when (número de periodos, importe de los pagos, capital inicial, capital final y momento de devengo de intereses).

La siguiente función está pensada para conocer el importe total de los intereses que voy a obtener en el depósito. Se denomina “INTERESES”, y requiere de los atributos rate, nper, pmt, pv y when (tipo de interés, número de periodos, importe de los pagos, capital inicial y momento de devengo de intereses).

Finalmente, la función denominada “TIEMPO” está pensada para conocer el tiempo que requiere mantener una inversión en un depósito, conociendo el tipo de interés, el pago, el capital inicial y el momento de devengo de los intereses (rate, pmt, pv, when).

Que se hayan definido estas funciones para los depósitos no implica que no pudiéramos hacer uso de ellas para otro tipo de productos como préstamos o hipotecas.

A continuación, vamos a crear nuestra clase Depósitos. Para ello debemos emplear la palabra al inicio `class` y a continuación el nombre.

Después debemos introducir nuestro método constructor mediante `def __init__(self, parámetros)`, para a continuación declarar los atributos. El método constructor no es imprescindible, pero si necesario para indicarle a Python que cuando se *instancia*<sup>3</sup> objeto debe asignarle los argumentos que hemos creado.

En nuestro caso además hemos creado una serie de propiedades que deben tener los atributos para evitar errores en los cálculos posteriores que debemos realizar. En nuestro caso hemos determinado que el argumento *nper* no puede ser inferior a 0 y que además deber ser un número entero, no una letra o número decimal. Por su parte el argumento *pv* sí que puede ser un número decimal, pero en cualquier caso siempre deberá ser mayor que 0 porque corresponde a la inversión que vamos a realizar. En cuanto al argumento *pmt* se ha determinado que puede ser igual a 0 pero nunca inferior, e igualmente puede contener decimales, al igual que el argumento *fv*. El tipo de interés por su parte (*rate*) igualmente debe ser un número positivo y siempre estará definido por decimales pues es importante indicarlo en tanto por uno a pesar de estar haciendo referencia a un porcentaje.

---

<sup>3</sup> Acción de crear un objeto a partir de una clase

Podemos ver parte del código desarrollado en la siguiente imagen:

```

2      @property
3      def nper(self): return self._nper
4      @nper.setter
5      def nper(self, nper):
6          if type(nper) == int and nper >= 0 or type(nper) == type(None):
7              self._nper = nper
8          else: print("La duración del periodo debe ser > 0")
9
10     @property
11     def pv(self): return self._pv
12     @pv.setter
13     def pv(self, pv):
14         if (type(pv) == int or type(pv) == float) and pv > 0:
15             self._pv = pv
16         else: print("La inversión inicial debe ser > 0")
17
18     @property |
19     def pmt(self): return self._pmt
20     @pmt.setter
21     def pmt(self, pmt):
22         if (type(pmt) == int or type(pmt) == float) and pmt > 0:
23             self._pmt = pmt
24         elif pmt == 0:
25             self._pmt = 1
26         else: print("Los pagos deben ser > 0 o iguales a 0")
27
28

```

A continuación, indicamos nuestro método constructor que debe contener todos los argumentos necesarios para realizar nuestros cálculos, en caso de no existir o desconocerlos se indicará un 0.

```

def __init__(self, rate, nper, pmt, pv, typ, fv = None):
    self.rate = rate
    self.nper = nper
    self.pmt = pmt
    self.pv = pv
    self.typ = typ
    self.fv = fv

```

Aunque hay valores que podemos desconocer, debemos tener presente qué información queremos obtener, y si tenemos los datos necesarios para llegar a ella. Por ejemplo, si queremos conocer los intereses, pero solo tenemos un capital inicial y un plazo, sin tipo de interés ni capital final no será posible averiguarlo. Pero si tenemos un capital inicial, un capital final y unos intereses, podemos averiguar la tasa empleada.

A continuación, añadimos las funciones que previamente habíamos definido, asignándoles los argumentos del método constructor, lo que nos permitirá realizar los cálculos sin necesidad de añadir nuevamente los datos, con una vez que se indiquen podremos realizar todas las operaciones que contiene nuestra clase depósitos.

```

def INTERESES(self):
    return INTERESES(self.rate, self.nper, self.pmt, self.pv, self.typ)

```

Y además añadimos nuevos métodos, que consisten en realizar una serie de cálculos y análisis que posteriormente no van a ayudar a tomar decisiones. Añadimos *AnalisisInter* que consiste en calcular el tipo de interés al que está remunerado nuestro depósito o *AnalisisFinal*, que nos realiza un resumen de la inversión realizada indicando el tiempo, el tipo de interés, los intereses y el valor final de la inversión.

```
#Este análisis nos permite conocer el tipo de interés después de haber obtenido nuestros intereses.
#O bien, buscar un depósito que esté remunerado a un determinado tipo de interés.
```

```
def AnalisisInter(self):
    Inter = round(RATE(self.nper, self.pmt, -self.pv, self.fv, self.typ)*100,2)
    return print("""Si invierte un capital de {} euros, durante {} años, y obtiene al final {} euros,
    Su inversión estará remunerada a un {} %""".format(self.pv, self.nper, self.fv, Inter))
```

Finalmente, al introducir los argumentos y llamar al método nos devolverá la información que necesitamos.

```
1 #Conocer los intereses finales.
2
3 depo1 = Depositos(0.07,4,0,1000,0, None)
4
5 depo1.AnalisisFinal()
```

```
Si invierte un capital de 1000 euros, durante 4 años, a un interés de 0.07,
obtendrá unos intereses de 306.36 euros, y un capital final de 1306.36 euros.
```

Pero donde realmente encontramos la ventaja de Python es al realizar el análisis no de un depósito, sino de varios, cuantos más depósitos tengamos que analizar más potente encontraremos esta herramienta.

Nos hemos inventado una serie de datos, y les hemos asignado nombres de entidades bancarias (podríamos haber elegido colores o cualquier otra opción), y los vamos a utilizar para realizar una comparativa y su gráfica.

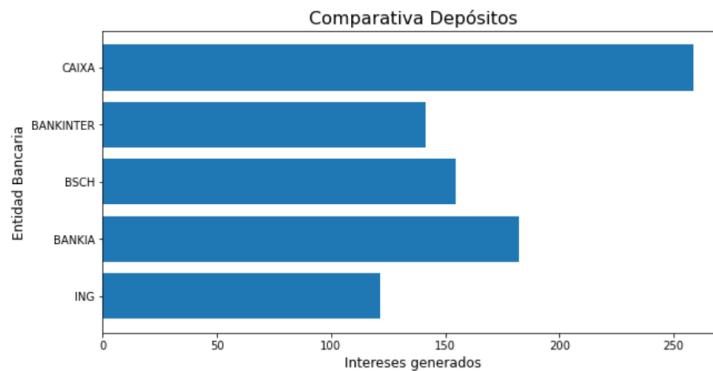
```
1 ING = Depositos(0.03, 4, 0, 1000, 0, fv = None)
2 BANKIA = Depositos(0.035, 5, 0, 1000, 0, None)
3 BSCH = Depositos(0.05, 3, 0, 1000, 0, None)
4 BANKINTER = Depositos(0.02, 7, 0, 1000, 0, None)
5 CAIXA = Depositos(0.04, 6, 0, 1000, 0, None)
```

A continuación, creamos una lista a partir de los intereses que generan cada uno de los depósitos de nuestro ejemplo.

```
1 Intereses1 = []
2 a = ING.INTERESES(), BANKIA.INTERESES(), BSCH.INTERESES(), BANKINTER.INTERESES(), CAIXA.INTERESES()
3 Intereses1.extend(a)
```

Este es el código que nos devuelve una gráfica:

```
1 p = ['ING', 'BANKIA', 'BSCH', 'BANKINTER', 'CAIXA']
2 plt.figure(figsize = (10, 5))
3 plt.barh(p, Intereses1 , height=0.8, align='center')
4 plt.title("Comparativa Depósitos", fontsize = 16)
5 plt.xlabel("Intereses generados", fontsize = 12)
6 plt.ylabel("Entidad Bancaria", fontsize=12)
7 plt.show()
```



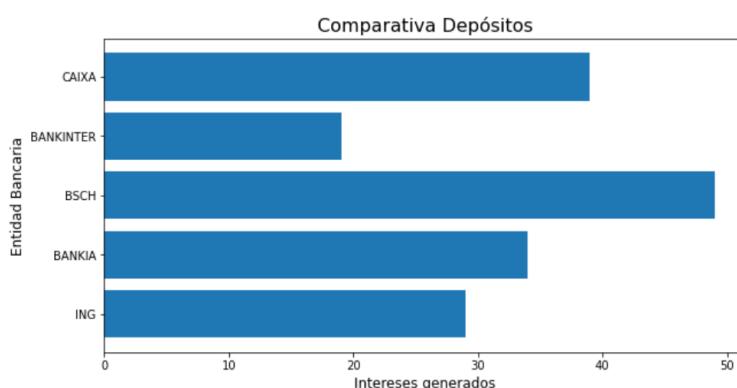
Como podemos ver en la gráfica, en función de los intereses que generan los depósitos, comprobamos que el ofrecido por La Caixa es el que más rentabilidad nos aporta. Si observamos con detenimiento los datos, podemos indicar que no son homogéneos. Cada depósito nos indica un periodo de inversión diferente, por lo que la cantidad de intereses puede derivarse de la cantidad de tiempo que se mantiene la inversión y no así de su rentabilidad. Para unificar la información hemos creado una pequeña variación, sustituyendo el tiempo por una variable denominada  $n$ .

```

1 n = 1
2 ING = Depositos(0.03, n, 0, 1000, 0, fv = None)
3 BANKIA = Depositos(0.035, n, 0, 1000, 0, None)
4 BSCH = Depositos(0.05, n, 0, 1000, 0, None)
5 BANKINTER = Depositos(0.02, n, 0, 1000, 0, None)
6 CAIXA = Depositos(0.04, n, 0, 1000, 0, None)

```

Al sustituir  $n$  por cualquier número, unificamos los períodos y podemos realizar un análisis más realista.



Aquí podemos comprobar como nuestra gráfica ha cambiado, y ahora es Banco Santander quien nos ofrece una mayor rentabilidad si mantenemos nuestra inversión la misma cantidad de períodos.

### 3.1.6 ANÁLISIS DE PROYECTOS / INVERSIONES

Uno de los objetivos de este trabajo, es lograr mostrar a Python como una herramienta útil para la persona dedicada a las Finanzas, ya sea un Controller Financiero de una empresa, un Asesor Financiero de una Entidad Bancaria o un Analista de Riesgos.

Una vez que hemos comprobado distintas maneras de realizar cálculos con Python, es decir, de crear distintas calculadoras, vamos a desarrollar el análisis de un proyecto, intentando parametrizar todas las variables, para poder introducir los datos que tengamos y obtener el Análisis de distintos proyectos.

Para ello, lo primero que vamos a plantear son las necesidades que tenemos a la hora de valorar un proyecto, ya sea la creación de una nueva empresa o la expansión de esta. Debemos tener presentes 3 aspectos importantes:

- Liquidez
- Rentabilidad
- Riesgo

Que elementos debemos tener para realizar cualquier análisis:

Los flujos de caja del proyecto o Cash Flows, ya que el beneficio o pérdida contable está afectado por una serie de normas contables que no tienen por qué reflejar la situación patrimonial real de la empresa. Normalmente el periodo que se utiliza es años, pudiendo ser mensual o trimestral en proyectos más cortos, y podemos realizar las proyecciones a futuro entre 3 y 7 años, reservando más años solo en aquellos proyectos en los que el periodo de maduración es elevado.

En los proyectos más simples como una inversión en bonos, el cálculo es sencillo, tenemos una inversión inicial negativa, para después tener una serie de ingresos hasta el último periodo donde obtendremos además el capital invertido. Sin embargo, el cálculo en empresas es algo más complicado, se parte del beneficio contable, y se realizan una serie de ajustes que nos darán lugar a Flujo de Caja Libre, independiente de la financiación que utilicemos para el proyecto.

Para el cálculo se tiene en cuenta los cobros y pagos, no los ingresos y gastos.

## MÉTODO

- (+/-) Resultado contable (eliminados los gastos financieros derivados del proyecto)  
(+) Amortizaciones y provisiones  
(-) Inversiones en activo inmovilizado  
(+) Desinversiones en activo inmovilizado  
(-) Aumento de necesidades operativas de fondos (NOF)  
(+) Disminución de necesidades operativas de fondos (NOF)
- 

(=) *Cash-Flow* (o Flujo de Caja Operativo Libre) del proyecto.

Las Necesidades Operativas de Fondos se calculan a partir de las existencias más los clientes menos los proveedores, con lo que obtenemos el volumen de recursos que la empresa debe dedicar a la producción y gestión comercial, descontando la financiación recibida de los proveedores.

## TÉCNICAS VALORACIÓN DE PROYECTOS

### *Liquidex*

- Payback o Periodo de Recuperación. - tiempo necesario para recuperar la inversión inicial

### *Rentabilidad*

- VAN y TIR, siendo la rentabilidad la capacidad del proyecto de generar rentas.

### *Riesgo*

- El riesgo de un proyecto es la incertidumbre asociada a las rentas futuras, que aumenta en cuanto el plazo que esperamos obtener rentas sea mayor. Para recoger el riesgo lo normal es utilizar un perfil conservador a la hora de calcular los flujos de caja, disminuyéndolos si es necesario.

Partes de un proyecto de inversión

- Desembolso inicial =  $A$
- Flujos de Caja =  $Q_1$
- Años =  $t$
- Duración del proyecto =  $n$

Podemos recoger estos elementos en este diagrama temporal, siendo el momento 0 el momento puntual en el que se realiza la inversión, no siendo ni un año ni un periodo.

**Payback.** es el tiempo necesario para que las entradas de caja generadas por nuestra inversión hasta el momento anulen o compensen las salidas que está originando.



Si los flujos de caja son positivos e iguales, entonces el periodo de recuperación vendrá dado por la división entre desembolso inicial.

$$\text{Payback} = \frac{A}{Q}$$

Son mejores aquellos proyectos en los que se recupera antes la inversión, siendo lo más habitual que se estime un límite, por ejemplo, no invertir en proyectos cuyo periodo de recuperación sea mayor de 3 años.

Desventajas:

- No tiene en cuenta los flujos de caja después de recuperar la inversión.
- No tiene en cuenta el paso del tiempo ni el valor del dinero en el tiempo, lo que se puede solucionar utilizando el Payback actualizado, que utiliza los flujos de caja actualizados al momento 0 a una determinada tasa de descuento.

Este método prioriza la liquidez de la inversión, lo que implica una aversión al riesgo, ya que la desconfianza de recuperar tiene como consecuencia que elija aquella que tarda menos tiempo.

Es un método muy sencillo de aplicar, y aunque carece de una fórmula concreta en Excel, hemos realizado una función en Python que permite conocer el retorno de una inversión a partir de los flujos de caja, teniendo en cuenta esta fórmula.

```

1 FC = [-2000, 200, 1500, 700, 100, 500] #Flujos de caja de un proyecto
2
3 Acum_FC = 0
4
5 for i in range(len(FC)):
6     Acum_FC += FC[i]
7
8     if Acum_FC > 0:
9         print("El Payback del Proyecto son {} años". format(i))
10        break
11
12 elif Acum_FC <= 0 and i == len(FC)-1:
13     print("El proyecto no recupera su inversión")

```

El Payback del Proyecto son 3 años

$$\text{Payback} = \frac{\text{Inversión Inicial}}{\text{Flujos de Caja}}$$

Esta función es un bucle que realiza la suma por año de los flujos de caja del proyecto, teniendo en cuenta que el primer año el flujo es negativo dado que se trata de la inversión inicial que debemos realizar. Si a lo largo de la vida del proyecto se tuvieran que hacer nuevas inversiones, estas serían siempre con signo negativo. Finalmente, cuando el saldo acumulado es mayor a 0 sabemos que hemos recuperado la inversión inicial y nos devuelve los años.

**VAN.-** Valor Actualizado Neto de una inversión es igual al valor actualizado de todos los flujos de caja esperados o lo que es lo mismo, la diferencia entre el valor actual de los cobros que se van a generar menos los pagos.

Para su cálculo necesitamos las siguientes variables:

- Desembolso inicial =  $A$
- Cobros esperados a final de cada período =  $C_t$
- Pagos esperados a final de cada período =  $P_t$
- Flujo Neto de Caja =  $Q_t = C_t - P_t$
- Duración del proyecto =  $n$
- Tipo de descuento o coste de capital de la empresa =  $k$

La tasa de descuento suele ser el coste de oportunidad de los recursos financieros o rentabilidad de una inversión alternativa, con el mismo nivel de riesgo que estamos analizando.

El cálculo del VAN se realiza:

$$VAN = -A + \frac{Q_1}{(1+K)} + \frac{Q_2}{(1+K)^2} + \dots + \frac{Q_n}{(1+K)^n} = -A + \sum_{t=1}^n \frac{Q_t}{(1+K)^t}$$

Se deben elegir aquellos proyectos que generen un VAN positivo, porque indica que el proyecto genera riqueza por encima de proyectos alternativos, siendo siempre mejor cuanto mayor sea este indicativo.

Sus ventajas:

- Considera la pérdida de valor del dinero en el tiempo.
- Cálculo sencillo

Desventajas:

- Lo difícil de este método es determinar la tasa adecuada.

Para ello debemos acudir al cálculo del Coste de los Recursos Propios (CAPM o Ke)

$$Ke = Rf + \beta_\mu(E_m - Rf)$$

Donde:

- Rendimiento libre de riesgo =  $Rf$
- Beta apalancada =  $\beta_\mu$
- Rendimiento esperado del mercado =  $E_m$

Posteriormente, en el apartado 3.1.9 se desarrolla en profundidad el cálculo de la Beta para empresas que cotizan en bolsa.

En muchas ocasiones se utiliza el tipo de interés vigente en el país para la deuda sin riesgo añadiendo una prima en función del perfil de riesgo del proyecto específico, es decir sustituimos la prima de riesgo de la compañía ( $\beta_\mu(E_m - Rf)$ ) por la prima de riesgo del proyecto.

En Excel disponemos de una fórmula que nos permite calcular el VAN de un proyecto, la podemos encontrar con el nombre en inglés *NPV* (*Net present value*) o *VAN*, y que además está presente en la librería Numpy financial de Python con el nombre en inglés *npv*, por lo que importando la librería podemos usarla para analizar nuestros proyectos. A continuación, podemos ver su funcionamiento, además de su comparativa desarrollando la fórmula sin la necesidad de emplear la librería:

```

3 cf=[-1000, 200, 300, 1000, 500]
4 r = 0.05
5 f = 1+r
6
7 VAN2 = npf.npv(rate=r, values=cf)
8 print(VAN2)

```

737.7738699410224

```

3 cf=[-1000, 200, 300, 1000, 500]
4 r = 0.05
5 f = 1+r
6
7 VAN = 0
8 for i in range(len(cf)):
9     VAN += cf[i]/f**(i)
10
11 print(VAN)

```

737.7738699410224

Hemos desarrollado dos maneras adicionales de calcular el VAN mediante Python, todas ellas válidas y que obtienen el mismo resultado, siendo quizás el más “pythonista” el último por su sencillez y simplicidad.

```

3 cf = np.array(cf)
4 r = 0.05
5 f = 1+r
6 n = np.array([0, 1, 2, 3, 4])
7
8 VA = cf / f**n
9 VAN = VA.sum()
10 print(VAN)

3 VAN = (cf/f**n).sum()
4 print(VAN)

```

737.7738699410224

Estos cálculos están pensados para proyectos que mantienen su tasa de interés constante a lo largo de la vida del proyecto o inversión, pero nos podemos encontrar con situaciones en las que el tipo de interés se va modificando a lo largo del tiempo. En estos casos, podemos considerar que el tipo de interés cambia a partir de ese momento, o bien que debemos valorar cada periodo a un tipo de interés de la siguiente forma:

$$VAN = -7.000 + \frac{1.000}{1.02} + \frac{1.200}{1.02 * 1.03} + \frac{1.400}{1.02 * 1.03 * 1.04} + \frac{1.600}{1.02 * 1.03 * 1.04 * 1.05} + \frac{1.800}{1.02 * 1.03 * 1.04 * 1.05 * 1.06} + \frac{2.000}{1.02 * 1.03 * 1.04 * 1.05 * 1.06 * 1.07}$$

Veamos un ejemplo de ambos casos

```

4 cf = [-7000]+list(range(1000, 2200, 200))
5 cf = np.array(cf)
6
7 print("Flujos de caja: {}".format(cf))
8
9 rates = np.arange(0.01,0.075,0.01)
10 factores = rates + 1
11 print("Tasas de descuento: {}".format(rates))
12
13 n = np.arange(0,7,1)
14 print("Periodos: {}".format(n))
15
16
17 VAN = (cf/factores**n).sum()
18 print("VAN usando la ETTI: {:.2f}".format(VAN))

1 cf = [-7000]
2 b = list(range(1000, 2200, 200))
3 cf1 = cf + b
4 cf = np.array(cf1)
5 print("Flujos de caja: {}".format(cf))
6
7 #Lista de tipos de interés entre 1% y 6,5% con incrementos del 1%
8 rates = np.append([],np.arange(0.01,0.075,0.01))
9 factores = rates + 1
10 print('Factores (1+r):', factores)
11
12 factoresMult=[] # Lista formada por los fm (factores multiplicados)
13 fm=1 # Los fm (factores multiplicados) se inicializan en 1
14 for f in factores:
15     fm*=f
16     factoresMult.append(fm)
17 print('factoresMult:',factoresMult)
18
19 VA = (cf/factoresMult).sum()
20 print('VA:',VA)

```

Flujos de caja: [-7000 1000 1200 1400 1600 1800 2000]  
Tasas de descuento: [0.01 0.02 0.03 0.04 0.05 0.06 0.07]  
Periodos: [0 1 2 3 4 5 6]  
VAN usando la ETTI: 350.18

Flujos de caja: [-7000 1000 1200 1400 1600 1800 2000]  
Factores (1+r): [1.01 1.02 1.03 1.04 1.05 1.06 1.07]  
factoresMult= [1.01, 1.0302, 1.061106, 1.1035502400000001, 1.158727752, 1.22

En el primer caso obtenemos un VAN de 350,18 frente a los 807,64 obtenidos mediante la segunda opción. Para evitar tener que realizar cálculos independientes, se ha creado una función con el argumento  $n$ , en caso querer usar el primer caso de cálculo tan solo hay que omitir este argumento, y en caso de querer usar la segunda forma del cálculo se ha de crear una lista igual al número de periodos del proyecto.

```

1 cf = np.array([-7000, 1000, 1200, 1400, 1600, 1800, 2000])
2 rate = np.array([1, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07])
3 n = [0, 1, 2, 3, 4, 5, 6]
4
5 def Calculo_Van(values, rate, n=None):
6     if not n:
7         a = 1
8         rate1 = []
9         for i in rate:
10            a = a*i
11            rate1.append(a)
12        |
13    VAN = (values/rate1).sum()
14    return VAN
15
16 else:
17     VAN = (values/(rate**n)).sum()
18     return VAN
19
20 print(Calculo_Van(values=cf, rate=rate, n=n))
21 print(Calculo_Van(values=cf, rate=rate))

```

350.1752684214696  
815.7245572726517

**TIR.**- nos indica la rentabilidad anual del proyecto, implícita en los flujos de caja del proyecto analizado. Este debe compararse con la tasa mínima que se quiera obtener, así como con el coste de financiación, siempre siendo preferible una TIR mayor.

Siendo K la rentabilidad mínima esperada:

$$\begin{aligned} TIR(r) > K &\Rightarrow \text{ACEPTAR PROYECTO} \\ TIR(r) < K &\Rightarrow \text{RECHAZAR PROYECTO} \end{aligned}$$

Sus ventajas respecto al VAN:

- Se expresa en porcentaje, por lo que es un concepto más comprensible
- No necesitamos tener un tipo de interés, ni conocer el coste de los recursos propios, aunque sí que debemos determinar nuestra rentabilidad mínima.

Sus desventajas:

- La fórmula matemática compleja que lleva aparejada.
- La posibilidad, escasa, que un proyecto tenga diversos TIR, o que sea negativo o no tenga un TIR real.

En cualquier caso, VAN y TIR se consideran complementarios a la hora de realizar un análisis:

$$\begin{aligned} \text{Si } r > K &\longrightarrow \text{VAN} > 0 && \text{ACEPTAR} \\ \text{Si } r < K &\longrightarrow \text{VAN} < 0 && \text{RECHAZAR} \end{aligned}$$

Afortunadamente Excel cuenta con una fórmula que nos permite calcular el TIR de una forma muy sencilla, podemos encontrarlo dentro de las fórmulas financieras con el nombre en inglés *IRR* o bien *TIR*.

Además, también podemos encontrar esta fórmula dentro de la librería Numpy finance con el nombre en inglés *IRR*. Tanto en Excel como en Python requieren de los mismos argumentos y funcionan exactamente igual. A continuación, podemos ver un ejemplo de su uso en Python.

```

3 cf=[-1000, 200, 300, 1000, 500]
4 r = 0.05
5
6 TIR = npf.irr(cf)
7 print(TIR)
8

```

0.28116785840687086

Hemos desarrollado otra versión, realizando una fórmula que busca el momento en el que nuestro VAN es igual a cero, momento que define nuestra TIR

```

6 paso = 0.000001
7 objetivo = 0
8 tolerancia = 0.001
9
10 while True:
11     f = 1 + r
12     VAN = 0
13     for i in range(len(cf)):
14         VAN += cf[i]/f**i
15     diff = VAN - objetivo
16
17     if abs(diff) > tolerancia:
18         if diff < 0:
19             r -= paso
20         elif diff > 0:
21             r += paso
22
23     else:
24         break
25
26 print(VAN, r)
27

```

-0.0002973604021008214 0.2811679999993661

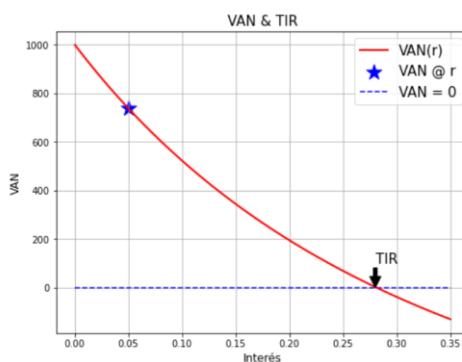
```

12 def TIR(cf):
13     k_inicial=0
14     k_final=1
15     a=npv(k_inicial,cf)
16     b=npv(k_final,cf)
17     if a==0:
18         return k_inicial
19     elif b==0:
20         return k_final
21     elif a*b>0:
22         print("En el rango considerado no se corta el eje")
23         return 'NaN'
24     else:
25         while True:
26             k_medio=(k_inicial+k_final)/2
27             m=npv(k_medio,cf)
28             if abs(m)<1e-12:
29                 break
30             else:
31                 if a*m<0:
32                     k_final=k_medio
33                 elif b*m<0:
34                     k_inicial=k_medio
35
36     return k_medio

```

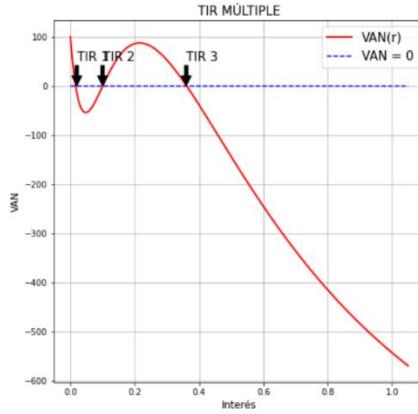
Este método como podemos apreciar mantiene pequeñas diferencias en el resultado. Por ello se buscó una fórmula que fuera más precisa, cuyo resultado es exactamente igual a 0.28116785...

Para representar gráficamente el concepto de VAN y TIR, hemos generado una lista de tipos de interés, y hemos calculado el VAN de nuestro proyecto a distintos tipos de interés. La gráfica representa un proyecto, por lo que a medida que aumenta la tasa, disminuye nuestra rentabilidad y en consecuencia el VAN. La línea roja del gráfico representa los VAN que se han ido generando a los distintos tipos de interés. La línea azul representa el VAN de nuestro proyecto igual a cero. En el momento que ambas líneas se cruzan vemos representado el TIR del proyecto. A un tipo de interés del 0.05, como es el del proyecto que estamos analizando, vemos representado el VAN mediante una estrella azul. Podemos representarlo gráficamente:



En ocasiones, algunos proyectos que requieren de inversiones a lo largo de la vida del proyecto, por ello podemos encontrarnos que su VAN se iguala a cero en distintos momentos y, por tanto, disponen una TIR múltiple. Podemos ver el siguiente ejemplo:

Podemos ver su representación gráfica:



**INDICE DE RENTABILIDAD.** – Sistema utilizado cuando tenemos varios proyectos y tenemos que elegir uno, queriendo saber cuál es el que aporta mayor valor a nuestra empresa, para lo que confeccionamos un ranking mediante la siguiente fórmula:

$$I.R. = \frac{VA}{A} = \frac{\sum_{t=1}^n \frac{Q_t}{(1+r)^t}}{A}$$

Aunque puede parecer complicada, lo único que tenemos que hacer es actualizar los flujos de caja sin tener en cuenta la inversión inicial, y dividirlo entre la inversión inicial. Es similar al payback, pero teniendo en cuenta el tiempo y actualizando los flujos de caja al momento presente.

En Excel no disponemos de ninguna fórmula que nos permita su cálculo, pero si lo hemos desarrollado en Python mediante el siguiente código:

```

1 FC = [-2000, 200, 1500, 700, 100, 500]
2 r = 0.05
3 II = []
4 FCP = []
5
6 for i in FC:
7     if i < 0:
8         II.append(i)
9     else:
10        FCP.append(i)
11
12 VA = npf.npv(r, FCP)
13
14 IR = VA / II
15 print(-IR)

[1.38061353]

```

Finalmente, al igual que creamos una clase para los depósitos, hemos desarrollado una clase para proyectos que nos permitan analizar diferentes opciones de una forma rápida y sencilla.

### 3.1.7 LETRAS DE CAMBIO, LETRAS DEL TESORO, BONOS Y OBLIGACIONES

Una **letra de cambio** es un documento mercantil mediante el cual una persona o empresa queda obligada a pagarle una cantidad de dinero a otra en un plazo preestablecido. En otras palabras, es un título-valor que garantiza una deuda entre dos partes. En toda letra de cambio intervienen como mínimo las siguientes partes:

- *El librador o girador.* Es la persona (física o jurídica) que emite la letra de cambio, es decir, quién elabora el documento y da la orden de pago.
- *El librado o girado.* Es quién acepta la orden de pago y, por tanto, tiene la obligación de pagar la deuda a su beneficiario. El librado es el deudor oficial de esta operación.
- *El beneficiario o tomador.* Es a quién finalmente se le paga, es decir, la persona que recibe el dinero correspondiente a la deuda. Puede coincidir con el librador, aunque no tiene por qué. Dependerá de si la letra ha sido endosada o no.

Si estamos en posesión de letras de cambio, podemos llevarlas al banco para su descuento, y aquí es donde entra la parte de matemática financiera para calcular los intereses que nos cobrará el banco, y el montante final que nos será abonado.

La fórmula que nos permite calcular el montante final, así como los intereses es:

$$D = \frac{C_n * d * n}{B}$$

Donde:

*Cn* = nominal del título

*n*= número de días que van desde la fecha de liquidación a la fecha de vencimiento

*d*= tipo o tasa de descuento

*B*= base anual aplicada (360 o 365)

Podemos ver un ejemplo práctico aplicando Python:

**Ejemplo 2.** El 7 de enero de 2013 una empresa descuenta en Gestión Bank una letra de cambio de 2.485 euros de valor nominal y cuya fecha de vencimiento es el 7 de marzo de 2014. El banco le aplica un tipo de interés de descuento del 6 %.

¿Cuál será el interés total cobrado por el banco en la operación de descuento?

```
In [88]: 1 Cn = 2485
2 f_liq = date(2013,1,7)
3 f_vto = date(2014,3,7)
4 t = 0.06
5 b = 360
6 n = f_vto -f_liq
7
8
9 D = (Cn*(n.days)*t)/b
10 print("Los intereses totales cobrados por el banco son {}".format(round(D,2)))
11
12 efectivo = Cn-D
13 print("Y por tanto el efectivo recibido es ", round(efectivo,2))
```

Los intereses totales cobrados por el banco son 175.61  
Y por tanto el efectivo recibido es 2309.39

Dentro del módulo financiero de Excel, existe la fórmula *INT.ACUM.V* que nos permite realizar el cálculo de forma sencilla. Aunque Numpy financial tiene numerosas fórmulas financieras al igual que Excel, no es el caso de esta, por lo que hemos creado una función que funcione de manera similar, utilizable en otro tipo de productos como los pagarés.

```
1 def INT_ACUM_V(i, f_vto, f_liq, b, Cn):
2     ...
3     INTERESES ACUMULADOS
4     =====
5
6     nos devuelve el total de intereses producidos por un activo financiero con pago de dichos
7     intereses al vencimiento
8
9     i = tasa
10    f_vto = fecha de vencimiento real de letra utilizando el formato date(AAAA,MM,DD)
11    f_liq = fecha en la que llevamos la letra al descuento, formato date(AAAA,MM,DD)
12    b = base de referencia según tablas
13    Cn = importe de la letra de cambio
14
15    TABLA BASES
16    1 = 365
17    2 = 360'''
18
19    if b == 1:
20        b = 365
21    elif b == 2:
22        b= 360
23    else:
24        print("Error! Debe introducir 1 o 2 para el valor b")
25
26    n = f_vto -f_liq
27    D = (Cn*(n.days)*i)/b
28    return print("Los intereses totales cobrados por el banco son {}".format(round(D,2)))
```

Para conocer el importe en efectivo que se va a recibir al llevar a descuento una letra o pagaré en Excel disponemos de *Precio\_descuento*, basada en la fórmula matemática:

$$P = VR \cdot \left(1 - \frac{d \cdot r}{B}\right), \text{ donde:}$$

*P*: es el precio del activo o título.

*VR*: es el valor de amortización o reembolso del título en la fecha de vencimiento.

*n*: es el número de días que van desde la fecha de liquidación del título hasta su vencimiento.

*d*: es la tasa de descuento

*B*: base que se toma para el cálculo (360 o 365 días)

En España debemos calcular el tipo de interés que es equivalente a la tasa de descuento que emplea la fórmula:  $i = \frac{d}{1-n \cdot \frac{i}{360}}$ , donde:

$n$ : son los días que van desde la fecha de adquisición hasta la fecha de vencimiento.

$i$ : es el tipo de interés

$d$ : es la tasa de descuento

Como no hemos encontrado esta función en ninguna de las librerías de Python, la hemos creado para que funcione de forma similar a la que contiene Excel.

```

1 def PRECIO_DESCUENTO(f_liq, f_vto, descuento, amortizacion, base):
2     """
3         CÁLCULO DE TOTAL EFECTIVO RECIBIDO
4         =====
5
6         Esta función calcula el precio expresado en tanto por ciento del nominal
7         de cualquier activo descontado a una determinada tasa de descuento.
8
9         liquidación = Introducimos la fecha de descuento del título
10        vencimiento = Introduciremos la fecha de vencimiento del título descontado
11        descuento = Es la tasa de descuento anual
12        amortizacion = Es el valor de reembolso en tanto por ciento del título a la fecha de vencimiento (Nominal)"""
13
14    if base ==1:
15        base = 365
16    elif base == 2:
17        base = 360
18    else:
19        print("Error! Indique 1 para bases de 365 días o 2 para bases de 360 días.")
20
21    n = f_vto - f_liq
22    #print(n.days)
23    i = descuento/(1-(n.days*descuento/360))
24    #print(i)
25    inte = amortizacion*i/base*n.days
26    #print(inte)
27    P = amortizacion - (amortizacion*i/base*n.days)
28    return round(P,2)
```

Para conocer el tipo de interés o el rendimiento, Excel dispone de la fórmula *RENDTO.DESC*, basada también en una fórmula matemática:  $P = \frac{VR}{1+\frac{n \cdot i}{B}}$  que nos permite

despejar el interés obteniendo la fórmula:  $i = \frac{(VR-P)*B}{P*n}$ .

Dado que tampoco hemos encontrado una función dentro de las librerías de Python que se asemeje a esta, hemos desarrollado una con funcionamiento similar.

```

1 def RENDTO_DESC(liquidacion,vencimiento,pr,amortizacion,base):
2     """
3         CÁLCULO DE TIPO INTERÉS
4         =====
5
6         Esta función nos calcula el tipo de interés o rendimiento,
7         en régimen de capitalización simple (año comercial de 360 días),
8         de un activo financiero emitido al descuento y siempre que conozcamos su valor de reembolso
9         o amortización en su fecha de vencimiento.
10
11        liquidación = Introducimos la fecha de compra o adquisición del activo financiero
12        vencimiento = Introduciremos la fecha de vencimiento del activo financiero
13        pr = Es el precio de la letra o pagaré expresado en tanto por ciento de su valor nominal
14        amortizacion = Es el valor de reembolso expresado en tanto por ciento del nominal del título"""
15
16    if base == 1:
17        base = 365
18        n = vencimiento - liquidacion
19        n = n.days
20    elif base == 2:
21        base = 360
22        n = vencimiento - liquidacion
23        n = n.days
24    elif base == 3:
25        year1 = liquidacion.year
26        month1 = liquidacion.month
27        day1 = liquidacion.day
28        year2 = vencimiento.year
29        month2 = vencimiento.month
30        day2 = vencimiento.day
31        n = calcular_dias(year1, month1, day1, year2, month2, day2)
32        base = 360
33    else:
34        print("Error! Indique un número correcto para indicar la base.")
35
36
37    inter = pr - amortizacion
38    #print(inter)
39    #print(n)
40    i = (inter * base) / (n*amortizacion)
41    #Una vez obtenido el interés, hay que convertirlo.
42    tasa = i/(1+n*i/base)
43
44    #el resultado está en tanto por 1.
45
46    return round(tasa,5)
47
48

```

Las **Letras del Tesoro** son títulos de deuda pública emitidos al descuento como forma de financiación del Estado, normalmente a corto plazo, y aunque son inversiones de renta fija no están exentas de cierto riesgo, aunque menor en comparación con otros productos, por eso su rentabilidad suele ser menor también. Que las letras estén emitidas al descuento significa que el valor nominal de la letra es de 1.000 euros, y su precio de compra estará por debajo de ese valor, la diferencia entre el valor de compra y venta nos determinará su rentabilidad. Por tanto, las letras del tesoro pueden ser adquiridas en múltiplos de 1.000 euros, y dado que son a corto plazo (hecho que las distingue de los Bonos y Obligaciones). Actualmente se emiten a 3, 6, 9 y 12 meses.

Mientras que las Letras del Tesoro están pensadas como un producto a corto plazo (menos de 18 meses), los **bonos** se emiten a largo plazo (entre 3 y 5 años) y las **obligaciones** a más largo plazo (30 ó 50 años), por tanto, funcionan de una forma parecida a los depósitos, solo que la emisión de deuda se hace en múltiplos de 1.000 euros. Además, la contratación de Bonos u Obligaciones no se realiza a descuento, sino que se ofrece un tipo de interés al que será remunerado anualmente la inversión, es decir, si contratas un bono a 3 años, cada año se abonarán unos intereses a una rentabilidad previamente fijada, se denomina **Cupón**. Por tanto, su funcionamiento es similar a los depósitos, solo que los intereses están calculados mediante interés simple.

En referencia a las Letras, Excel dispone de dos fórmulas financieras que son *LETRA.DE.TES.RENDTO* y *LETRA.DE.TES.PRECIO* que tampoco aparecen en las librerías consultadas de Python, por lo que se han creado en Python

```

1 def LETRA_DE_TES_RENDTO(Pm, f_adq, f_vto):
2     ...
3     CÁLCULO DEL RENDIMIENTO
4     =====
5
6     Esta función nos calcula el tipo de interés o rendimiento,
7     en régimen de capitalización simple (año comercial de 360 días),
8     de una Letra del Tesoro Público o pagaré a un plazo que no supere los 365 días
9     y siempre que conozcamos su precio expresado en porcentaje de su valor nominal.
10
11    Pm = Es el precio de la letra o pagaré expresado en tanto por ciento de su valor nominal
12    f_adq = Introducimos la fecha de compra o adquisición del activo financiero
13    f_vto = Introduciremos la fecha de vencimiento del activo financiero'''
14
15
16    n = f_vto - f_adq
17
18    dias=365
19    if n.days > dias:
20        print(""""
21                    ERROR!! LAS LETRAS DE CAMBIO NO PUEDEN SER SUPERIORES A UN AÑO""")
22    else:
23        i = ((100-Pm)*360)/(Pm*n.days)
24        i = i*100
25
26    return round(i,3)

1 def LETRA_DE_TES_PRECIO(i, f_adq, f_vto):
2     ...
3     CÁLCULO DEL PRECIO
4     =====
5
6     Esta función nos calcula el precio por cada cien euros de valor nominal,
7     de una Letra del Tesoro o cualquier otro activo financiero que funcione al descuento,
8     tales como pagarés o letras de cambio, a un tipo o una tasa de descuento determinada,
9     siempre que los días de descuento sean inferiores a 366.
10
11    i = Tipo o tasa de descuento anual del título
12    f_adq = Introducimos la fecha de compra o adquisición del activo financiero
13    f_vto = Introduciremos la fecha de vencimiento del activo financiero
14
15    *Nota* Esta función es muy similar a PRECIO_RENDERIMIENTO, solo que las Letras del Tesoro están limitadas
16    a vencimientos de 3 / 6 o 12 meses, por lo que períodos superiores indica un error.''''
17
18    n = f_adq - f_vto
19    dias=365
20    if n.days > dias:
21        print(""""
22                    ERROR!! LAS LETRAS DE CAMBIO NO PUEDEN SER SUPERIORES A UN AÑO""")
23    else:
24        d = i/(1+(n.days*i/360))
25        Po = 100*(1+(d*n.days/360))
26
27    return round(Po,3)
```

### 3.1.8 PRÉSTAMOS

El préstamo es una operación financiera, en la que un banco o entidad entrega una cantidad de dinero a una empresa, persona o entidad, que deberá ser devuelto en un plazo determinado. La persona o entidad que presta se denomina prestamista, y la persona o entidad que recibe el dinero se denomina prestatario. Por esta operación se cobrarán una serie de intereses, que tendrá que abonar el prestatario al prestamista.

Las partes que componen un préstamo son: el capital o principal, que es la cantidad de dinero prestada. El término amortizativo o pago, la cantidad que es amortizada cada

periodo (mensual, trimestral, semestral o anual) haciendo que la deuda disminuya. Los intereses, calculados sobre la parte del capital o deuda pendiente. El capital vivo, es la parte de la deuda pendiente de devolver. El capital amortizado, que es la deuda que ya ha sido devuelta. Y por último la cuota, que es la suma del término amortizativo y los intereses.

Existen diferentes métodos para calcular la amortización del préstamo, que influyen tanto en el coste financiero de la operación como en las cuotas.

*Método Americano.* – consiste en amortizar el total de la deuda al vencimiento del préstamo, liquidando únicamente intereses en los períodos preestablecidos. Es el que menor coste tiene implícito, pero se ha de estar preparado para atender el total de la deuda. La cuota estará formada por los intereses, que serán constantes al igual que el capital vivo, dado que no se realizan amortizaciones. Para un préstamo de 100.000 euros a devolver en 5 períodos a un interés del 7% podemos ver el siguiente cuadro:

Periodo	Anualidad	Intereses	Amortización	Capital Vivo	Capital Amortizado
0	0	0	0	100000	0
1	7000	7000	0	100000	0
2	7000	7000	0	100000	0
3	7000	7000	0	100000	0
4	7000	7000	0	100000	0
5	107000	7000	100000	0	100000

*Método Italiano.* – en este caso la cuota de amortización se mantiene constante, mientras los intereses son decrecientes, por lo que la cuota amortizativa decrece en progresión aritmética. En los primeros años liquida menos intereses que otras modalidades, pero aumenta en los últimos tramos. Si tomamos como referencia el ejemplo anterior su cuadro de amortización quedaría de la siguiente forma:

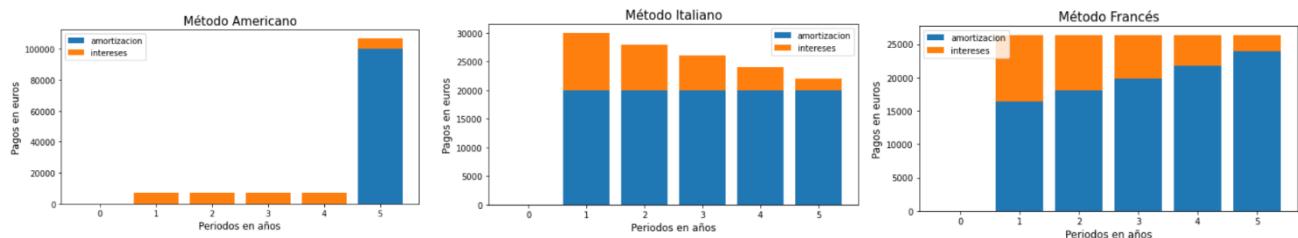
Periodo	Anualidad	Intereses	Amortización	Capital Vivo	Capital Amortizado
0	0	0	0	100000	0
1	27000	7000	20000	80000	20000
2	25600	5600	20000	60000	40000
3	24200	4200	20000	40000	60000
4	22800	2800	20000	20000	80000
5	21400	1400	20000	0	100000

*Método francés.* – quizás el más extendido, se caracteriza por tener cuotas constantes, siendo la amortización creciente en progresión geométrica, mientras los intereses son decrecientes. Al final de la vida del préstamo apenas tiene coste financiero, por lo que incentiva a realizar amortizaciones anticipadas al principio de la vida de este. Continuando con el ejemplo, podríamos obtener el siguiente cuadro de amortización:

Periodo	Anualidad	Intereses	Amortización	Capital Vivo	Capital Amortizado
0	0	0	0	100000	0
1	24389	7000	17389	82611	17389
2	24389	5783	18606	64005	35995
3	24389	4480	19909	44096	55904
4	24389	3087	21302	22794	77206
5	24389	1596	22794	0	100000

Como hemos podido comprobar, el método con mayor coste financiero es el americano dado que mantiene el mayor capital vivo durante más tiempo y por tanto nos genera más intereses.

En estas gráficas podemos valorar cómo van cambiando los intereses, la amortización y las cuotas en función del método elegido.



Para realizar cálculos de préstamos, Excel dispone de fórmulas financieras, algunas de ellas ya las hemos visto en apartados anteriores, dado que sirven para el cálculo de cualquier inversión o financiación. Estas fórmulas son *vf*, *va*, *nper*, *tasa* o *pago*. Sin embargo, disponemos de otras fórmulas como *pagoint* (*ipmt* en inglés), o *pagoprinc* (*ppmt* en inglés), que nos pueden servir de gran ayuda a la hora de calcular el pago de intereses por periodos o la cuota de amortización.

Estas fórmulas se encuentran incluidas en la librería de Python, Numpy financial con los nombres en inglés (*ipmt* y *ppmt*). Son de especial importancia en los préstamos que utilizan el método francés para el cálculo de amortización, ya que nos permite saber las cuotas mensuales, así como la parte que corresponde a intereses y la parte que corresponde a amortización. A continuación, podemos ver su funcionamiento:

```

1 #en primer lugar vamos a calcular la cuota constante
2
3 C0 = 100000
4 n = 5
5 tasa = 0.07
6
7 pago = npf.pmt(rate=tasa, nper=n, pv=-C0, fv=0, when='end')
8 print(round(pago,2))

```

24389.07

En este caso, la fórmula *pmt* requiere que nuestro valor final del préstamo sea 0, dado que no queremos dejar deuda pendiente. Normalmente el devengo de intereses, así como la amortización se realizan al final del periodo, pero podría darse el caso que se realizar al comienzo, y tal caso habría que dejarlo indicado.

```

1 #Vamos a calcular el capital pendiente de cada periodo
2
3 pendiente = [C0]
4
5 for i in range(1,n+1,1):
6     c = npf.fv(rate=tasa, nper = i, pmt = pago, pv= -C0)
7     pendiente.append(c)
8
9
10 print(pendiente)

```

[100000, 82610.93055586258, 64004.626250635585, 44095.880644042685, 22793.522844988227, -0.0]

Mediante el bucle *for* podemos calcular el capital vivo o pendiente de cada uno de los períodos, utilizando la fórmula *fv*, e indicando el pago que se ha calculado previamente. El capital vivo de un préstamo a un instante corresponde a la reserva matemática por la derecha de ese instante, una vez se ha amortizado el término que

vence en ese momento. Existen tres métodos para calcular la reserva matemática, recurrente, retrospectivo y prospectivo, siendo el más usado en los préstamos de tip francés el método prospectivo.

```

1 #Calculamos la amortización de cada año
2
3 amortz = [0]
4
5 for i in range(1,n+1,1):
6     b = npf.pmt(rate=tasa, per=i, nper=n, pv=-C0, fv=0, when='end')
7     amortz.append(b)
8 print(amortz)
9
10
[0, 17389.069444137396, 18606.304305227015, 19908.745606592904, 21302.357799054407, 22793.52284498822]
```

Aquí podemos comprobar como la fórmula  $ppmt$  nos permite conocer la parte de la cuota que es destinada a la amortización del préstamo, disminuyendo la deuda en progresión geométrica.

```

1 #Calculamos los intereses de cada periodo
2 inter = [0]
3
4 for i in range(1,n+1,1):
5     d = npf.ipmt(rate=tasa, per=i, nper=n, pv=-C0, fv=0, when='end')
6     inter.append(d)
7
8 inter
9
[0,
array(7000.),
array(5782.76513891),
array(4480.32383754),
array(3086.71164508),
array(1595.54659915)]
```

Finalmente, mediante la fórmula  $ipmt$  calculamos la parte de nuestra cuota periódica destinada al pago de intereses de nuestro préstamo. Mediante una sencilla suma, podemos comprobar que cada periodo nos da como resultado el pago inicial que hemos calculado.

A partir de aquí podemos construir nuestro cuadro de amortización, en este caso usando la librería *tabulate*, incluida en Python. Esta librería nos permitirá dar la apariencia de un cuadro, y nos facilitará la lectura de los datos, mediante el siguiente código:

```

1 #Hacemos un cuadro de amortización para ver los datos con la Librería tabulate
2 datos = []
3 saldo = C0
4 saldo2 = 0
5 linea1 = [0,0,0,0,C0,0]
6 datos.append(linea1)
7
8 Anualidad = npf.pmt(rate=tasa, nper=n, pv=-C0, fv=0, when='end')
9
10 for i in range(1, n+1):
11     pago_capital = npf.ppmt(rate=tasa, per=i, nper=n, pv=-C0, fv=0, when='end')
12     pago_int = Anualidad - pago_capital
13     saldo -= pago_capital
14     saldo2 += pago_capital
15
16     linea = [i, format(Anualidad, '.0f'), format(pago_int, '.0f'),
17               format(pago_capital, '.0f'), format(saldo, '.0f'), format(saldo2, '.0f')]
18
19     datos.append(linea)
20
21 print(tab.tabulate(datos, headers=['Periodo', 'Anualidad', 'Intereses',
22                                     'Amortización','Capital Vivo','Capital Amortizado'],
23                     tablefmt = 'psql'))
```

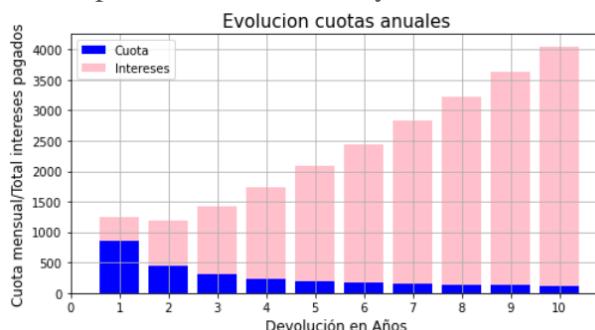
Periodo	Anualidad	Intereses	Amortización	Capital Vivo	Capital Amortizado
0	0	0	0	100000	0
1	24389	7000	17389	82611	17389
2	24389	5783	18606	64005	35995
3	24389	4480	19909	44096	55904
4	24389	3087	21302	22794	77206
5	24389	1596	22794	0	100000

Una de las mayores ventajas de Python es la rapidez de cálculo en grandes volúmenes de datos, y comprobarlo hemos realizado un ejercicio ficticio, generando 100 tipos de interés diferentes. Posteriormente hemos calculado la cuota, los intereses y total pagado para un préstamo de 100.000 euros a devolver en 10 años. Mediante la librería *pandas* hemos creado un DataFrame que funciona de manera similar a una tabla de Excel, y que nos permite realizar análisis de forma rápida y sencilla.

	round(Prestamos2.describe(),2)			
	Interés	Pago mensual	Total pagado	Total intereses pagados
count	100.00	100.00	100.00	100.00
mean	0.05	12981.42	129814.21	29814.21
std	0.03	1820.72	18207.16	18207.16
min	0.00	10000.00	100000.00	0.00
25%	0.02	11411.14	114111.36	14111.36
50%	0.05	12919.05	129190.49	29190.49
75%	0.07	14518.75	145187.54	45187.54
max	0.10	16204.69	162046.92	62046.92

En este caso, si nos fijamos en la columna de los intereses pagados, que nos va a indicar el coste financiero de los préstamos, vemos que el gasto medio es de 29.814,21 €, para un tipo de interés medio del 5%.

De igual manera podemos estudiar la diferencia entre pagar un préstamo a más o menos plazo, estableciendo los costes financieros correspondientes y tomando la decisión del tiempo más apropiado que permita ajustar cuota e intereses. En el siguiente gráfico se ha analizado un préstamo de 10.000 euros a un tipo de interés del 7% para devolver entre 1 y 10 años.



Mediante el comando describe, obtenemos en segundos las principales medidas estadísticas. Así sabemos que tenemos en nuestra tabla 100 valores, su media, su desviación típica, el mínimo, el máximo, así como los principales cuartiles.

Podemos comprobar que si bien la cuota desciende de forma brusca el si devolvemos el préstamo en dos años, apenas varía en cambio entre 9 y 10 años. Sin embargo, vemos como los intereses aumentan en proporción al número de años que tardamos en devolver el préstamo.

Una de las ventajas que nos ofrece trabajar con DataFrames y la librería *pandas* es la posibilidad de guardar nuestros datos en Excel, de forma automática mediante un código sencillo “*NombreDataFrame.to\_excel("Nombrearchivo.xlsx", sheet\_name="Sheet1")*”, automáticamente tendremos un archivo de Excel con toda la información.

A	B	C	D	E	F	G
1	Nº de cuotas	Cuota Mensual	Total pagado	Total intereses pagados	Variación cuota %	Variación inter %
2	0	12	85,60748179	1027,289781	27,28978146	0
3	1	24	43,87138973	1052,913354	52,91335362	-48,75285569
4	2	36	29,9708971	1078,952296	78,95229577	-31,68464166
5	3	48	23,02929357	1105,406091	105,4060914	-23,16114699
6	4	60	18,87123364	1132,274019	132,2740186	-18,0555253
7	5	72	16,10493266	1159,555152	159,5551516	-14,6588243
8	6	84	14,13390907	1187,248362	187,248362	-12,23863291
9	7	96	12,65992001	1215,352321	215,3523212	-10,42874305
10	8	108	11,51777217	1242,865507	212,8655072	-8,025702501

### 3.1.9 EXTRACCIÓN DE DATOS WEB Y CÁLCULO DE BETAS

Anteriormente, hemos visto como se utiliza la Beta de una compañía para calcular el coste de los recursos propios de una empresa, tasa habitual que se usa para comparar con el VAN de un proyecto y valorar si ese proyecto es rentable o por el contrario es preferible invertir en otro proyecto o inversión.

En el caso de empresas que cotizan en bolsa, podemos tener esa información disponible en páginas o informes oficiales, como por ejemplo la Comisión Nacional de Valores en España, u otros organismos de carácter oficial que publican informes periódicamente. Sin embargo, dado que son empresas que cotizan en bolsa, podemos acceder al histórico de sus cotizaciones y realizar el cálculo. Para ello, extraemos los datos de un tiempo determinado (normalmente un año), tanto de las cotizaciones diarias de la empresa como la cotización del mercado. Posteriormente calculamos el retorno de la inversión de un día para otro ( $100\% - (\text{Cotz}(n) / \text{Cotz}(n-1))$ ), esto nos va a permitir calcular la covarianza del título, así como la varianza del mercado. Esto nos permite aplicar la fórmula de la Beta:

$$\beta = \frac{\text{Covarianza } \sigma(\text{título mercado})}{\text{Varianza } \sigma^2(\text{mercado})}$$

En los casos de empresas que no cotizan en bolsa, se ha de realizar por semejanza mediante el Método de Hamada. Para ello se deben buscar empresas que trabajen en el mismo sector o parecidos, con características similares que coticen en bolsa. Una vez tenemos las betas de las empresas cotizadas, debemos desapalancar las betas:

$$\beta_{\mu} = \frac{\beta_L}{1 + \frac{\text{Deuda Financiera (1 + tipo impositivo)}}{\text{Fondos Propios}}}$$

Para realizar el cálculo de la media ponderada de las Betas desapalancadas. Una vez obtenida la media ponderada, debemos volver a apalancar con los datos de la empresa de la que queremos saber la beta (la empresa no cotizada)

$$\beta_L = \beta_{\mu} * \left(1 + \frac{\text{Deuda Financiera (1 + tipo impositivo)}}{\text{Fondos Propios}}\right)$$

Mediante Python podemos obtener los datos de cotización de una manera sencilla y rápida. Como hemos visto, tenemos disponibles multitud de librerías que nos hacen el trabajo más sencillo, y para la lectura de datos financieros, la librería *yfinance* y *pandas* podemos extraer los datos de la página *yahoo.finance*, donde se publican las cotizaciones de distintos mercados financieros, entre ellos el IBEX35.

```
1 Ibex_empresas = pd.read_html('https://es.finance.yahoo.com/quote/%5EIBEX/components/')[0]
2 Ibex_empresas
```

Su código es sencillo, tan solo tenemos que indicar *pd* (referente a la librería *pandas*), seguido de un punto *read* (común a la lectura de todo tipo de archivos), seguido de *\_html*, esto lo que indica es la lectura de una página web. De esta página nos interesa saber los símbolos con los que se publica la información financiera, dado que esta información nos la publica con el nombre de la empresa, sino con unas abreviaturas.

Símbolo	Nombre de la empresa	Último precio	Cambio	Cambio de %	Volumen
0 ENG.MC	Enagás, S.A.	1910	1	+0,05%	545.330
1 NTGY.MC	Naturgy Energy Group, S.A.	2158	2	+0,09%	808.505
2 AENA.MC	Aena S.M.E., S.A.	14235	-15	-0,11%	175.147

Con los símbolos generamos una lista para extraer los datos de cotización diarios por el periodo de tiempo que necesitemos, incluyendo los datos del mercado.

```
| 1 simbolos = Ibex_empresas['Símbolo'].tolist() + ['^IBEX']
 2 data = yf.download(simbolos, period='5y', auto_adjust=True)
[*****100%*****] 31 of 31 completed
```

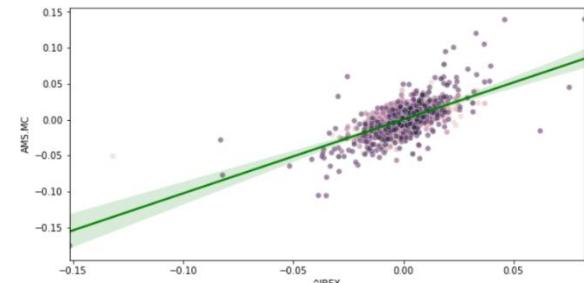
Este código nos extrae el precio de apertura, la cotización máxima, la cotización mínima, el volumen de operaciones y la cotización de cierre por fechas de los 30 valores del IBEX, así como los del propio IBEX35. Para el cálculo que estamos buscando nos quedaremos con la columna “Close” que nos marca el precio de cierre.

```
1 renta = np.log(1 + data.loc[:, 'Close'].pct_change()).dropna(how = 'all')
2 var = renta['^IBEX'].var()
3 beta = renta.cov()/var
4 beta['^IBEX'].head(31).to_frame('Beta').T
```

	ACS.MC	ACX.MC	AENA.MC	AMS.MC	ANA.MC	BBVA.MC	BKT.MC	CABK.MC	CLNX.MC	COL.MC ..
<b>Beta</b>	1.18363	0.876446	0.941887	1.010228	0.836426	1.470858	1.112739	1.328242	0.505796	0.796473 ..

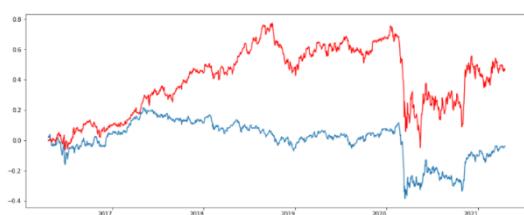
En cuestión de segundos hemos podido extraer la información de las cotizaciones de 30 empresas durante 5 años, y calcular sus betas. Incluso, mediante la librería de gráficos *matplotlib*, podemos ver la relación entre una compañía y el mercado, en este caso hemos escogido Amadeus, con una beta prácticamente neutra, y que por tanto sigue la tendencia del mercado.

```
1 def plot_regression(renta, asset):
2     plt.figure(figsize=(10,5))
3     sns.scatterplot('^IBEX', asset, data=renta,
4                     hue=renta.index.year, alpha=0.6, legend=False)
5     sns.regplot('^IBEX', asset, data=renta, scatter=False, color='green')
6
7 asset = 'AMS.MC'
8 plot_regression(renta, asset)
```

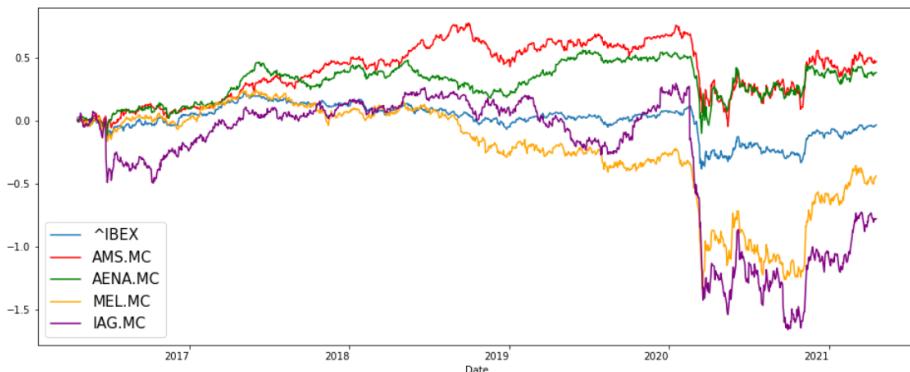


En este otro gráfico podemos comprobar como ambos siguen la misma tendencia:

```
1 renta['^IBEX'].cumsum().plot(subplots=True, figsize=(16,7))
2 renta[asset].cumsum().plot(subplots=True, figsize=(16,7), color = 'red')
```



Incluso podemos ver el históricos de varias empresas a la vez:



Finalmente, en este apartado hemos creado una función que nos permita el cálculo de la Beta de una empresa, de forma sencilla:

```

1 def beta(valor,periodo,mercado):      #asignamos las variables
2     simbol = [valor, mercado]          #creamos una lista para después extraer los datos de la página de yahoo
3     data = yf.download(simbol, period=periodo, auto_adjust=True) #descargamos los datos que necesitamos
4     renta = np.log(1 + data.loc[:, 'Close'].pct_change()).dropna(how = 'all') #calculamos la rentabilidad
5     var = renta[(mercado)].var() #calculamos la varianza del mercado
6     cov = renta.cov()           #calculamos la covarianza del valor
7     beta = renta.cov()/var      #calculamos la Beta de la compañía
8     return beta[mercado].head(1).to_frame('Beta').T #Finalmente mostramos el dato que necesitamos.
9

1 valor = 'IAG.MC' #recordemos que debemos indicar el simbolo de la empresa.
2 periodo = '5y'   #podemos seleccionar varios periodos. (1d,5d,1mo,3mo,6mo,1y,2y,5y,10y,ytd,max)
3 mercado = '^IBEX'
4
5 beta(valor, periodo, mercado)

[*****100*****] 2 of 2 completed

IAG.MC
Beta 1.664964

```

### 3.1.10 AUTOMATIZACIÓN DE INFORMES.

Otra de las herramientas muy potentes de Python es la capacidad de automatizar tareas, así podemos realizar informes automáticamente, enviar correos electrónicos, descargar información de páginas webs y guardarlos en archivos de Excel para luego trabajar con ellos mediante otras herramientas. Existen multitud de librerías que nos pueden ayudar en este sentido, en este trabajo hemos utilizado *docx* además de *pandas*, para realizar un informe automático que podemos ver completo en los anexos.

Una de las funciones más útiles es la posibilidad de guardar gráficos como imágenes que posteriormente podemos incorporar en nuestros informes. Mediante el comando *plt.savefig*, guardaremos el archivo en formato jpg.

```

1 plt.figure(figsize=(6,10))
2 sector.plot(kind='pie', subplots=True, fontsize=12, ylabel='', colormap ='mako')
3 plt.title("Sectores mas influyentes en el IBEX35", fontsize=20)
4
5 plt.savefig("Tbex.jpg", bbox_inches='tight') #nos guarda la imagen
6
7 plt.show()

```

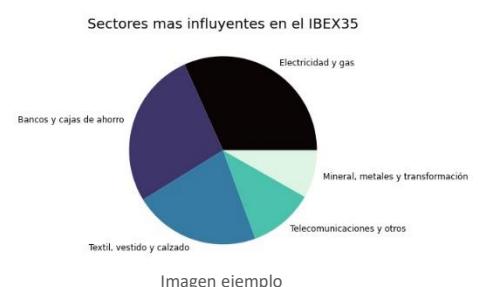


Imagen ejemplo

Hay un detalle importante que debemos tener en cuenta para obtener un buen resultado, que consiste en guardar la imagen previa a la visualización de la gráfica, al que corresponde el comando `plt.show()`.

También en este apartado se ha utilizado la librería pandas, incluida en el módulo de Python, a la que simplemente hay que llamar para disponer de muchas funciones predefinidas, en este caso las correspondientes a cálculos estadísticos. La función `mean` nos permite conocer la media aritmética, `cov` nos devuelve la covarianza o mediante `var` podemos conocer la varianza. La variedad de funciones de pandas te ofrece múltiples posibilidades, desde generar tablas dinámicas mediante la función `pivot_table`, hasta asignar índices o seleccionar columna o filas, eliminar valores nulos, unir varios archivos, renombrar columnas o filas, ordenar de mayor a menor y viceversa, en resumen, una amplia gama de funcionalidades con las que poder generar análisis de datos, así como gráficas o cálculos.

Para escribir documentos de Word de forma automática, en primer lugar se ha de crear el documento mediante el comando `Document()`, y a partir de ahí, se pueden escribir títulos mediante el comando `add_heading`, indicando los distintos niveles. Mediante el comando `add_paragraph`, se añaden los párrafos necesarios.

```

1 document = Document()
2 document.add_heading("ANÁLISIS DE IBEX_35", level=0)
3 document.add_heading("1. Características del Ibex35", level=1)
4 document.add_paragraph("El Ibex se compone de las 35 empresas con")
5 document.add_paragraph("Si comprobamos la distribución por sector")

```

Ejemplo de código y documento

## ANÁLISIS DE IBEX\_35

### 1. Características del Ibex35

El Ibex se compone de las 35 empresas con más licencia española que está formado por las bolsas de Madrid y utiliza como referencia para conocer la situación d

Si comprobamos la distribución por sectores, podemos ver que el sector más representativo, ya que supone un 22%

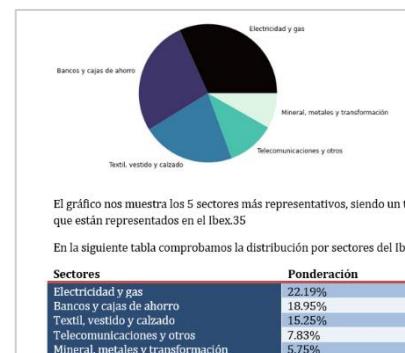
Para incluir fotografías, el comando que debemos utilizar es `add_picture`, y nos inserta las imágenes que deseemos. Así mismo podemos insertar tablas, mediante el comando `add_table`, pudiendo incluir datos calculados previamente.

```

6
7 document.add_picture("Ibex.jpg", width=Cm(11))
8
9 document.add_paragraph("El gráfico nos muestra los 5")
10 document.add_paragraph("En la siguiente tabla comprobaremos la")
11
12 table = document.add_table(rows=1, cols=2, style='Table')
13 #font = row_cells[0].text
14 table.height = Pt(8)
15
16 table.rows[0].cells[0].text = 'Sectores'
17 table.rows[0].cells[1].text = 'Ponderación'
18
19 for i,p in zip(sectores, ponderaciones):
20     row_cells = table.add_row().cells
21     row_cells[0].text = i
22     row_cells[1].text = str(round(p,2)) + '%'
23

```

Ejemplo de código y documento



Por último, no debemos olvidar guardar nuestro documento mediante el código `document.save("NombreArchivo.docx")` para posteriormente presentarlo, enviarlo por correo electrónico o cualquier otra tarea.

## 4 CONCLUSIONES

A lo largo de este trabajo hemos visto la potencia de Python a la hora de crear modelos, realizar cálculos y otro tipo de operaciones, que la convierten en una herramienta muy potente para emplear en el financiero, donde el volumen así como la frecuencia de las operaciones ha crecido, esta herramienta nos permite crear métodos para recopilar, procesar y analizar datos de una forma muy eficiente, optimizando los recursos disponibles.

Hoy en día, Python es ampliamente utilizado en sectores como los seguros o la gestión de inversiones, permitiendo crear modelos financieros, gestionar los riesgos o gestión de precios de activos. Su atractivo radica en la sintaxis simple que utiliza, en un ecosistema en el que podemos encontrar numerosas herramientas que nos facilitan las tareas, su fácil integración, así como su eficiencia y productividad.

Pero no solo en estos sectores es útil Python. En cualquier departamento financiero de cualquier empresa, grande o pequeña, pública o privada, se han de tomar decisiones para maximizar o minimizar distintas variables, como el beneficio, los ingresos, el rendimiento o costes. Cualquier herramienta que permita simplificar procesos, así como automatizar tareas, será de gran utilidad en estos departamentos de manera que se pueda emplear el tiempo en el análisis, y no tanto en la tarea en sí. Igualmente es importante tratar adecuadamente los datos, de manera que reduzcamos la posibilidad de errores en los cálculos, estimaciones y proyecciones.

Durante este trabajo hemos visto parte de las librerías más importantes, como Pandas, Numpy, Numpy Financial, Matplotlib... repasando las funciones comunes con Excel o mostrando la forma de crearlas si no existen. Hemos generado clases y funciones para realizar comparativas y análisis. Hemos creado gráficos que nos han ayudado a las explicaciones, y aportan soluciones visuales a los problemas que debemos analizar. Hemos extraído gran volumen de datos de una web, usándolos posteriormente para realizar cálculos y análisis, para finalmente crear un informe en Word que podríamos enviar a nuestro departamento, con todo lujo de detalles y gráficos incorporados. Por todo ello, queda demostrada la utilidad de Python en nuestro día a día, como herramienta para potenciar nuestros recursos.

## 5 ANEXOS

Para elaborar este trabajo ha sido necesario la elaboración de diversos códigos que nos permitieran crear paso a paso las funciones necesarias, si bien se han incluido en la parte del trabajo las que se han considerado más importantes que apoyaran las argumentaciones del trabajo, son muchos códigos los que no se han podido incluir pero que pueden resultar interesantes si presenta más inquietud por conocer el código empleado. Se han clasificado en función del capítulo que ocupan para mantener el mismo orden.

### 5.1.1 interés simple

Código para gráfica en interés simple

```

1 fig, ax = plt.subplots()
2 ax.bar(tiempo, intereses, color = "darkblue")
3 plt.title("Intereses en I. Simple", fontsize = 20)
4 plt.xlabel("Periodos", fontsize = 15)
5 plt.ylabel("Intereses en €", fontsize = 15)
6 plt.xticks(np.arange(tiempo[0], tiempo[-1]+1, 1))
7 plt.yticks(np.arange(0,intereses[-1]+1,100))
8 plt.show()

```

### 5.1.2 interés compuesto

Código de calculadoras financieras para interés compuesto:

```

1 def CalculaFinanciera():
2
3     mes = ['m', 'mes', 'mensual', 'uno']
4     año = ['a', 'anual', 'año']
5     trimestral = ['t', 'trimestral', 'trimestre', 'tres']
6     semestral = ['s', 'semestre', 'semestral', 'seis']
7     try:
8         tipo = input('Indique que tipo de periodo:
9                         m = mensual
10                        a = anual
11                        t = trimestral
12                        s = semestral '').lower()
13         capitalInicial = float(input('Importe a invertir: '))
14         tantoNominal = float(input('Introduzca el tipo de interés Nominal anual (Ejemplo 8%: 0.08) : '))
15         if tipo in mes:
16             periodoTemporal = 'Meses'
17             frecuencia = 12
18         elif tipo in año:
19             periodoTemporal = 'Años'
20             frecuencia = 1
21         elif tipo in trimestral:
22             periodoTemporal = 'Trimestres'
23             frecuencia = 4
24         elif tipo in semestral:
25             periodoTemporal = 'Semestres'
26             frecuencia = 2
27         periodos = float(input('Número de periodos (en {}): '.format(periodoTemporal)))
28         vf = capitalInicial * (1 + tantoNominal)**(periodos / frecuencia)
29         print("Su inversión tendrá un valor futuro de: {}".format(round(vf,2)))
30     except ValueError:
31         print('Error!, solo se admiten numeros')
32
33
34
35
36
37
38 CalculaFinanciera()

```

```

1 def calculo_compuesta(C0, i, n):
2     Cn=C0*(1+i)**n
3     return round(Cn,2)
4
5 print("""
6             El tipo de interés (i) y el número de periodos (n) que introduzca deben tener la misma unidad temporal.
7             Si el tipo de interés es anual, entonces el número de periodos han de ser años.
8             Si se trabaja con meses, entonces el tipo de interés introducido ha de ser un tipo mensual.
9
10            Por ejemplo, si el tipo de interés es del 12% anual y la operación de inversión a interés simple
11            dura 18 meses, indicaremos que el tipo de interés es i=0,12 y el número de años es n=1,5.
12            """)
13 C0 = float(input('Importe a invertir: '))
14 i = float(input('Introduzca el tipo de interés (Ejemplo 8%: 0.08) : '))
15 n = float(input('Número de periodos: '))
16
17 print('')
18 print('El capital final obtenido es', calculo_simple(C0, i, n))

```

El tipo de interés (i) y el número de periodos (n) que introduzca deben tener la misma unidad temporal.  
Si el tipo de interés es anual, entonces el número de periodos han de ser años.  
Si se trabaja con meses, entonces el tipo de interés introducido ha de ser un tipo mensual.

Por ejemplo, si el tipo de interés es del 12% anual y la operación de inversión a interés simple dura 18 meses, indicaremos que el tipo de interés es i=0,12 y el número de años es n=1,5.

Importe a invertir: 10000  
Introduzca el tipo de interés (Ejemplo 8%: 0.08) : 0.10  
Número de periodos: 4

El capital final obtenido es 14641.0

## Código para gráfica en interés compuesto

```

1 fig, ax = plt.subplots()
2 ax.bar(tiempo, intereses, color = "Darkblue")
3 plt.title("Intereses en I. Compuesto", fontsize = 20)
4 plt.xlabel("Periodos", fontsize = 15)
5 plt.ylabel("Intereses en €", fontsize = 15)
6 plt.xticks(np.arange(tiempo[0], tiempo[-1]+1, 1))
7 plt.yticks(np.arange(0,intereses[-1],100))
8 plt.show()

```

## Código para gráfica comparativa interés simple vs compuesto

```

1 fig, ax = plt.subplots()
2 plt.plot(intcomp, color = "blue", marker = '*', markersize = 4, label = "Intereses Compuestos")
3 plt.plot(intsim, color = "orange", marker = '*', markersize = 4, label = "Intereses Simples")
4 plt.title("Comparación Intereses Simples vs Compuestos", fontsize = 20)
5 plt.xlabel("Periodos", fontsize = 15)
6 plt.ylabel("Intereses en €", fontsize = 15)
7 plt.xticks(np.arange(tiempo[0], tiempo[-1], 1))
8 plt.yticks(np.arange(0,intcomp[-1],100))
9 plt.legend(loc = "best")
10 plt.show()

```

### 5.1.3 Uso de VF, VA, Tasa y Nper

#### Fórmula VF

**Capital Final**

```

1 i = float(input("Interes: "))
2 n = float(input("Plazo: "))
3 c0 = float(input("Capital inicial: "))
4
5 fv = npf.fv(rate=i, nper=n, pmt= 0,pv=-c0, when='end')
6 print("El capital final en esta operación es {}".format(round(fv,2)))

```

Interés: 0.10  
Plazo: 4  
Capital inicial: 10000  
El capital final en esta operación es 14641.0

```

1 c0 = float(input("Capital inicial: "))
2 n = float(input("Plazo: "))
3 Cn = float(input("Capital final: "))
4
5 i = ((Cn/c0)**(1/n))-1
6 print("El interés de esta operación es {} %".format(round(i,2)*100))

```

Capital inicial: 10000  
Plazo: 4  
Capital final: 14641  
El interés de esta operación es 10.0 %

#### Fórmula Tasa

**Interés**

Equivalencia con fórmula Excel **TASA** (Español) **RATE** (Inglés)

Funcionan igual, requiere los mismos parámetros (plazo, pago, capital inicial, capital final y al final o principio del periodo).

```

1 Cn = float(input("Capital final: "))
2 n = float(input("Plazo: "))
3 c0 = float(input("Capital inicial: "))
4
5 rate = npf.rate(nper=n, pmt=0, pv=-c0, fv=Cn, when='end')
6 print("El tipo de interés de esta operación es {}%".format(round(rate*100,2)))

```

Capital final: 14641  
Plazo: 4  
Capital inicial: 10000  
El tipo de interés de esta operación es 10.0 %

```

1 c0 = float(input("Capital inicial: "))
2 i = float(input("Interes: "))
3 n = float(input("Plazo: "))
4
5 Cn = c0*(1+i)**n
6 print("El Valor Final es {}".format(round(Cn,2)))

```

Capital inicial: 10000  
Interes: 0.10  
Plazo: 4  
El Valor Final es 14641.0

#### Fórmula VA

**Capital Inicial**

Equivalencia con fórmula Excel **VA** (Español) **PV** (Inglés)

Igualmente nos solicita exactamente los mismos parámetros (tasa, plazo, pago, capital final, final o principio del periodo)

```

1 Cn = float(input("Capital final: "))
2 n = float(input("Plazo: "))
3 i = float(input("Interés: "))
4
5 c0 = npf.pv(rate=i, nper=n, pmt=0, fv=-Cn, when='end')
6 print("El Capital Inicial de esta operación es {}".format(round(c0,2)))

```

Capital final: 14641  
Plazo: 4  
Interés: 0.10  
El Capital Inicial de esta operación es 10000.0

```

1 i = float(input("Interes: "))
2 n = float(input("Plazo: "))
3 Cn = float(input("Capital final: "))
4
5 c0 = Cn*(1+i)**(-n)
6 print("El capital inicial de esta operación es: {}".format(round(c0,0)))

```

Interes: 0.10  
Plazo: 4  
Capital final: 14641  
El capital inicial de esta operación es: 10000.0

#### Fórmula Nper

**Tiempo**

Equivalencia con fórmula Excel **NPER** (Español e Inglés)

También nos requieren los mismos parámetros que en Excel, tasa, pago, capital inicial, capital final y principio o final del periodo.

```

1 Cn = float(input("Capital final: "))
2 c0 = float(input("Capital inicial: "))
3 i = float(input("Interés: "))
4
5 n = npf.nper(rate=i, pmt=0, pv=c0, fv=-Cn, when='end')
6 print("El Plazo de esta operación es {}".format(n))

```

Capital final: 14641  
Capital inicial: 10000  
Interés: 0.10  
El Plazo de esta operación es 3.9999999999999964

```

1 c0 = float(input("Capital inicial: "))
2 i = float(input("Interes: "))
3 Cn = float(input("Capital final: "))
4
5 n = (math.log(Cn/c0))/(math.log(1+i))
6 print("El plazo de esta inversión es {} periodos".format(round(n,0)))

```

Capital inicial: 10000  
Interes: 0.10  
Capital final: 14641  
El plazo de esta inversión es 4.0 periodos

### Ejercicios prácticos:

5. Ana tiene otra preocupación. Cuando se jubile dentro de 25 años, quiere haber ahorrado 100.000 euros para poder retirarse tranquilamente en su pueblo. Ana quiere saber cuánto tiene que ahorrar anualmente para llegar a dicha cantidad, si invierte en un producto financiero que le asegura una rentabilidad del 3%

```

1 fv = 100000
2 n=25
3 #pmt =
4 i = 0.03
5
6
7 pmt = npf.pmt(rate=i, nper=n, pv=0, fv=-fv, when='end')
8 print("Ana tendrá que ahorrar {:.2f} euros anuales".format(pmt))

```

Ana tendrá que ahorrar 2,742.79 euros anuales

6. Eva ha decidido ahorrar cuando cumpla 30 años, haciendo aportaciones en un plan de pensiones que tiene una rentabilidad del 4%. Con lo ahorrado, cuando cumpla 65 años quiere obtener una mensualidad de 2.500 € al inicio de cada mes, y el importe restante se mantiene a un 3%, hasta que cumpla 90 años momento en el que quiere cobrar 100.000 €. ¿Cuánto tendrá que aportar anualmente Eva a su plan de pensiones?

```

1 #Primera fase del ejercicio: Calcular el capital inicial que debemos tener cuando cumpla Eva 65 años.
2
3 fv = 100000
4 n = 25      #12 meses * 25 años
5 i = 0.03    #3% / 12 meses
6 pmt = 2500
7 m = 12
8
9 pv = npf.pv(rate=i/m, nper=n*m, pmt=-pmt, fv=-fv, when='begin')
10 print("Eva necesita tener {:.2f} euros cuando cumpla 65 años".format(pv))
11
12 #Segunda fase del ejercicio: Calcular las aportaciones al plan de pensiones anualmente.
13
14 fv2 = pv
15 n2 = 35     #12 meses * 35 años
16 i2 = 0.04   #4% / 12 meses
17 pv1 = 0
18 m = 12
19
20 pmt = npf.pmt(rate = i2/m, nper = n2*m, pv= pv1, fv=-fv2)
21 print("Eva necesita ahorrar {:.2f} euros al mes para cumplir su objetivo".format(pmt))

```

Eva necesita tener 575,790.00 euros cuando cumpla 65 años

Eva necesita ahorrar 630.15 euros al mes para cumplir su objetivo

### 5.1.4 Rentas

Código para cálculo de rentas.

```

1 n = 120
2 pmt = 1000
3 i = 0.07
4 m = 12
5 im = (1+i)**(1/m)-1
6
7 #Método 1. Mediante fórmula de fv
8
9 fv = npf.fv(rate=im, nper=n, pmt=-pmt, pv=0, when='begin')
10 print("Resultado método 1: {} euros.".format(round(fv,2)))
11
12 #Método 2. Mediante fórmula Cf = pmt*(1+im)*(1+im)^n-1/im
13
14 Cf = pmt*(1+im)*((1+im)**n-1)/im
15 print("Resultado método 2: {} euros".format(round(Cf,2)))
16

```

Resultado método 1: 172018.88 euros.  
 Resultado método 2: 172018.88 euros

### Ejemplo práctico

3. Un agricultor desea adquirir un tractor dentro de 4 años por importe de 60.000 €. Para ello decide ahorrar efectuando aportaciones trimestrales de 3.000 €, comenzando hoy mismo la primera. El depósito se remunera al 6% nominal anual. Es consciente de que a pesar de las 16 aportaciones realizadas al fondo, no llegará al montante necesario y tendrá que pagar, dentro de 4 años, un importe X para poder adquirir el tractor. Calcular X.

```

1 n = 4
2 m = n*4
3 fv = 60000
4 pmt = 3000
5 i = 0.06
6 itr = i/4
7 # X =
8
9 fv1 = npf.fv(rate=itr, nper=m, pmt=-pmt, pv=0, when='begin')
10 print("Al final del periodo, el agricultor ha ahorrado {} euros".format(round(fv1,2)))
11
12 x = fv - fv1
13 print("El agricultor tendrá que abonar {} euros dentro de 4 años".format(round(x,2)))

```

Al final del periodo, el agricultor ha ahorrado 54604.07 euros

El agricultor tendrá que abonar 5395.93 euros dentro de 4 años

Código para gráfico de flechas.

```

1 fig, ax = plt.subplots()
2 ax.annotate('100 €', xy=(4, 1), xycoords='data',
3             xytext=(1, 1), textcoords='data',
4             arrowprops=dict(facecolor='black', shrink= 1),
5             horizontalalignment='right', verticalalignment='top')
6 ax.annotate("133,1 €", [4, 1])
7 ax.annotate('100 €', xy=(4, 2), xycoords='data',
8             xytext=(2, 2), textcoords='data',
9             arrowprops=dict(facecolor='blue', shrink= 1),
10            horizontalalignment='right', verticalalignment='top')
11 ax.annotate("121 €", [4, 2])
12 ax.annotate('100 ', xy=(4, 3), xycoords='data',
13             xytext=(3, 3), textcoords='data',
14             arrowprops=dict(facecolor='green', shrink= 1),
15             horizontalalignment='right', verticalalignment='top')
16 ax.annotate("110 €", [4, 3])
17 ax.annotate('100 €', xy=(4, 4), xycoords='data',
18             xytext=(4, 4), textcoords='data',
19             arrowprops=dict(facecolor='darkblue', shrink= 1),
20             horizontalalignment='right', verticalalignment='top')
21
22 plt.title("Valor final de las Rentas = 464,1", fontsize = 20)
23 plt.xlabel("Periodos", fontsize = 15)
24 plt.ylabel("Rentas recibidas", fontsize = 15)
25 plt.xlim(0, 5)
26 plt.xticks(np.arange(0, 5, 1))
27 plt.yticks([1, 5])
28 plt.show()
29

```

### 5.1.5 Depósitos

Código gráfico de depósitos

```

1 p = ['ING', 'BANKIA', 'BSCH', 'BANKINTER', 'CAIXA']
2 plt.figure(figsize = (10, 5))
3 plt.barh(p, Intereses1, height=0.8, align='center')
4 plt.title("Comparativa Depósitos", fontsize = 16)
5 plt.xlabel("Intereses generados", fontsize = 12)
6 plt.ylabel("Entidad Bancaria", fontsize=12)
7 plt.show()

```

### 5.1.6 VAN & TIR

Código para gráfico VAN - TIR

```

1 cf=[-1000, 200, 300, 1000, 500]
2 r = 0.05
3
4 VAN = 0
5 for i in range(len(cf)):
6     VAN += cf[i]/(1+r)**(i)      #Generamos el VAN de nuestro proyecto
7
8 L = list(range(0,351))           #Vamos a generar una Lista de supuestos tipos de interés, va poder ver como cambia nuestro
9
10 rs = []
11 for i in L:
12     rs.append(i/1000)
13
14 vans = []                      #Calculamos el VAN para cada uno de los intereses generados en la anterior lista.
15 for i in rs:
16     van = 0
17     for j in range(len(cf)):
18         van += cf[j]/(1+i)**(j)
19     vans.append(van)
20

```

```

1 plt.figure(figsize = (8, 6))
2 plt.scatter(x = r, y = VAN, s = 300, c="blue", marker = '*', label = "VAN @ r")
3 plt.plot(rs, vans, color = "red", linewidth = 2, linestyle='-', label = "VAN(r)")
4 plt.grid()
5 plt.hlines(y=0, xmin = rs[0], xmax=rs[-1], linestyle="dashed", color="blue", label="VAN = 0")
6 plt.title("VAN & TIR", fontsize=15)
7 plt.xlabel("Interés", fontsize=12)
8 plt.ylabel("VAN", fontsize=12)
9 plt.annotate("TIR", xy = (0.280, 0), xytext=(0.280, 100), arrowprops = {'color':'black'}, fontsize=15)
10 plt.legend(loc="best", fontsize=15)
11 plt.show()
12

```

## Ejemplos prácticos VAN & TIR

2. Aplicando una tasa de 10% efectivo anual, calcular el VAN de una operación de inversión cuyo desembolso es de 70.000 € y consta de 8 recuperaciones semestrales de 12.000 € cada una. En este caso, dado que la renta es constante podemos aplicar tanto la fórmula de VNA o VA, vamos a verlo aplicado.

```

1 #Fórmula VNA
2
3 cf = [-70000, 12000, 12000, 12000, 12000, 12000, 12000, 12000]
4 i = 0.10
5 m = 2
6 i2 = (1+i)**(1/m)-1 # i2 es el tanto semestral efectivo
7
8 VAN = npf.npv(rate = i2, values = cf)
9 print("{:.2f} €".format(VAN))
10
11 #Fórmula VA
12
13 pago = 70000
14 pmt = 12000
15 n = 8
16
17 VA = npf.pv(rate = i2, nper = n, pmt=-pmt, fv=0, when='end')-pago #tenemos que sacar fuera el pago inicial
18 print("{:.2f} €".format(VA))
19
7,933.38 €
7,933.38 €

```

5. Calcular la TIR de una operación de inversión con las siguientes características: Desembolso de 100.000 €. Siendo las recuperaciones de 10.000 € mensuales durante un año, salvo el 3º mes que no se recupera nada y el mes 9º donde se perciben 20.000 €.

```

1 cf = [-100000, 10000, 10000, 0, 10000, 10000, 10000, 10000, 20000, 10000, 10000, 10000]
2 m = 12 # frecuencia
3
4 r12 = npf.irr(cf)      #r12 es la TIR mensual
5 print('Rentabilidad mensual:{}% tanto efectivo mensual'.format(r12*100))
6 r = (1+r12)**m-1
7 print('TIR = {}% tanto efectivo anual'.format(r*100))

```

Rentabilidad mensual:2.6973355461856086% tanto efectivo mensual  
TIR = 37.62905031900152% tanto efectivo anual

## Código gráfica TIR Múltiple

```

1 plt.figure(figsize = (8, 8))
2
3 plt.plot(r, VAN, color = "red", linewidth = 2, linestyle='-', label = "VAN(r)")
4 plt.grid()
5 plt.hlines(y=0, xmin = r[0], xmax=r[-1], linestyle="dashed", color="blue", label="VAN = 0")
6 plt.title("TIR MÚLTIPLE", fontsize=15)
7 plt.xlabel("Interés", fontsize=12)
8 plt.ylabel("VAN", fontsize=12)
9 plt.annotate("TIR 1", xy = (0.020, 0), xytext=(0.020, 50), arrowprops = {'color':'black'}, fontsize=15)
10 plt.annotate("TIR 2", xy = (0.100, 0), xytext=(0.100, 50), arrowprops = {'color':'black'}, fontsize=15)
11 plt.annotate("TIR 3", xy = (0.360, 0), xytext=(0.360, 50), arrowprops = {'color':'black'}, fontsize=15)
12 plt.legend(loc="best", fontsize=15)
13 plt.show()

```

## Class de Proyectos. Código de funciones previas

```

1 #Damos forma a determinadas funciones que después vamos a incorporar en nuestra class
2
3 def VAN(rate, values):      #para su calculo necesitamos una tasa de descuento y unos flujos de caja
4     VAN = 0
5     for i in range(len(values)):
6         VAN += values[i]/(1+rate)**(i)
7     return VAN
8
9 def TIR(values):           #Para el cálculo de la TIR solo necesitamos los flujos de caja
10    TIR = npf.irr(values)
11    return TIR
12
13 def PAYBACK(values):       #Igualmente, para el Payback solo necesitamos los flujos de caja
14
15     Acum_FC = 0
16
17     for i in range(len(FC)):
18         Acum_FC += FC[i]
19
20     if Acum_FC > 0:
21         return i
22         break
23
24     elif Acum_FC <= 0 and i == len(FC)-1:
25         print("El proyecto no recupera su inversión")
26
27
28 def IR(rate, pv, values):  #Para el índice de rentabilidad necesitamos varios datos.
29     values = values[1:]
30
31     VA = npf.npv(rate, values)
32     IR = VA / pv
33     return IR
34

```

## Propiedades y atributos. Init de Class

```

1
2
3 #Definimos class Proyecto, empezando por las propiedades de las variables, evitando valores nulos.
4
5 class Proyecto:
6
7     @property
8     def nper(self): return self._nper #Todo proyecto debe tener un periodo superior a 0
9     @nper.setter
10    def nper(self, nper):
11        if type(nper) == int and nper > 0:
12            self._nper = nper
13        else: print("La duración del periodo debe ser > 0")
14
15     @property
16     def pv(self): return self._pv #Todo proyecto debe tener una inversión inicial superior a 0
17     @pv.setter
18     def pv(self, pv):
19         if (type(pv) == int or type(pv) == float) and pv > 0:
20             self._pv = pv
21         else: print("La inversión inicial debe ser > 0")
22
23     @property
24     def rate(self): return self._rate #En este caso, vamos a pedir una tasa de interés, más adelante veremos calcular
25     @rate.setter
26     def rate(self, rate):
27         if type(rate) == float and rate > 0:
28             self._rate = rate
29         else: print("El interés (8% = 0.08) de ser > 0")
30
31     def __init__(self, rate, nper, pv, values): #Definimos qué datos necesitamos para analizar nuestro proyecto
32         self.rate = rate                      #tasa de descuento
33         self.nper = nper                      #duración del proyecto
34         self.pv = pv                          #inversión inicial
35         self.values = values                 #flujos de caja del proyecto (más adelante veremos si podemos calcularlos)
36
37     def VAN(self):
38         return VAN(self.rate, self.values)   # definimos la función para todos los objetos de Proyecto.
39
40     def TIR(self):
41         return TIR(self.values)           # definimos la función para todos los objetos de Proyecto
42
43     def PAYBACK(self):
44         return PAYBACK(self.values)       # definimos la función de Payback para Proyecto
45
46     def IR(self):
47         return IR(self.rate, self.pv, self.values) # definimos la función de Indice Rentabilidad para Proyecto
48
49     def ANALISIS(self):
50         return "Este proyecto tiene un VAN de {}, una TIR de {}, su PayBack es de {} años, y su indice de retorno es de {}"
51
52

```

## Aplicación

1 Proyecto1.PAYBACK()	El Payback del Proyecto son 3 años
1 Proyecto1.IR()	'El Indice de Rentabilidad es 1.8246625634380738.'
1 Proyecto1.ANALISIS()	'Este proyecto tiene un VAN de 737.77, una TIR de 0.28, su PayBack es de 3 años, y su indice de retorno es de 1.82'
ualmente, podemos tener tantos proyectos como queramos, y analizarlos todos	
1 Proyecto2 = Proyecto(0.05,3,1000,[-1000, 500, 500, 500])	
2 Proyecto3 = Proyecto(0.03,4,500,[-500, 200, 200, 200, 200])	
3 Proyecto4 = Proyecto(0.02,2,200,[-200, 500, 500])	
4 Proyecto5 = Proyecto(0.08,5,1500,[-1500, 400, 400, 400, 400, 400])	
5 Proyecto6 = Proyecto(0.03,3,400,[-400, 500, 500, 500])	
6 Proyecto7 = Proyecto(0.07,2,800,[-800, 600, 600])	
7 Proyecto8 = Proyecto(0.05,4,700,[-700, 400, 400, 400, 400])	
8 Proyecto9 = Proyecto(0.06,3,900,[-900, 800, 800, 800])	
1 Proyecto2.ANALISIS()	'Este proyecto tiene un VAN de 361.62, una TIR de 0.23, su PayBack es de 3 años, y su indice de retorno es de 1.43'
1 Proyecto3.ANALISIS()	'Este proyecto tiene un VAN de 243.42, una TIR de 0.22, su PayBack es de 3 años, y su indice de retorno es de 1.53'

### 5.1.7 Letras de cambio, letras del tesoro y bonos.

#### Ejemplos

```

1 #desarrollo para introducir datos manualmente
2
3 datos_vto = input('Introduzca la fecha de vencimiento del activo (forma YYYY-MM-DD)')
4 año, mes, dia = map(int, datos_vto.split('-'))
5 vencimiento = date(año, mes, dia)
6 datos_liq = input('Introduzca la fecha de liquidación del activo (forma YYYY-MM-DD)')
7 año, mes, dia = map(int, datos_liq.split('-'))
8 liquidacion = date(año, mes, dia)
9 base = float(input('''Introduza:
10             1 para base de 365 días
11             2 para base de 360 días
12             3 para meses de 30 días y año 360 '''))
13 Amortizacion = float(input('Introduzca el valor de reembolso del activo '))
14 pr = float(input('Introduzca el precio del activo '))
15
16 RENDTO_DESC(liquidacion,vencimiento,pr,amortizacion,base)

Introduzca la fecha de vencimiento del activo (forma YYYY-MM-DD)2013-03-07
Introduzca la fecha de liquidación del activo (forma YYYY-MM-DD)2013-01-07
Introduza:
    1 para base de 365 días
    2 para base de 360 días
    3 para meses de 30 días y año 360
Introduzca el valor de reembolso del activo2421.38
Introduzca el precio del activo2458
: 0.08939

```

```

1 #Introducción datos manuales
2
3
4 par = float(input('Introduzca el valor nominal del Bono '))
5 datos_emi = input('Introduzca la fecha de emisión del Bono (forma YYYY-MM-DD)')
6 año, mes, dia = map(int, datos_emi.split('-'))
7 f_emi = date(año, mes, dia)
8 datos_liq = input('Introduzca la fecha de liquidación del Bono (forma YYYY-MM-DD)')
9 año, mes, dia = map(int, datos_liq.split('-'))
10 f_liq = date(año, mes, dia)
11 tasa = float(input('Introduzca el tipo de interés (Ejemplo 8%: 0.08) : '))
12 frecuencia = float(input('''Introduzca la frecuencia:
13             Para años = 1
14             Para semestres = 2
15             Para trimestres = 4 '''))
16 base = float(input("Introduzca 1 para base 365 días, 2 para base 360 días"))
17
18 INT_ACUM(f_emi, f_liq, tasa, par, frecuencia, base)

Introduzca el valor nominal del Bono 1000
Introduzca la fecha de emisión del Bono (forma YYYY-MM-DD)2014-03-15
Introduzca la fecha de liquidación del Bono (forma YYYY-MM-DD)2016-06-19
Introduzca el tipo de interés (Ejemplo 8%: 0.08) : 0.0325
Introduzca la frecuencia:
    Para años = 1
    Para semestres = 2
    Para trimestres = 4 1
Intoduzca 1 para base 365 días, 2 para base 360 días
: 73.548

```

### 5.1.8 Préstamos

#### Código para elaborar cuadros de amortización

```

1 #Hacemos un cuadro de amortización para ver los datos con la librería tabulate
2 datos = []
3 saldo = C0
4 saldo2 = 0
5 linea1 = [0,0,0,0,C0,0]
6 datos.append(linea1)
7
8 Anualidad = npf.pmt(rate=tasa, nper=n, pv=-C0, fv=0, when='end')
9
10 for i in range(1, n+1):
11     pago_capital = npf.ppmt(rate=tasa, per=i, nper=n, pv=-C0, fv=0, when='end')
12     pago_int = Anualidad - pago_capital
13     saldo -= pago_capital
14     saldo2 += pago_capital
15
16     linea = [i, format(Anualidad, '.0f'), format(pago_int, '.0f'),
17               format(pago_capital, '.0f'), format(saldo, '.0f'), format(saldo2, '.0f')]
18
19     datos.append(linea)
20
21 print(tab.tabulate(datos, headers= ['Periodo', 'Anualidad', 'Intereses',
22                                         'Amortización', 'Capital Vivo', 'Capital Amortizado'],
23                                         tablefmt = 'psql'))
24

```

## Código para gráficos

```

1 plt.figure(figsize = (8, 4))
2 plt.bar(range(0, n+1 ), Amortz, label = 'amortizacion')
3 plt.bar(range(0, n+1 ), inter, bottom= Amortz, label='intereses')
4 plt.legend(fontsize = 10)
5 plt.title("Método Italiano", fontsize = 15)
6 plt.xlabel("Periodos en años", fontsize = 12)
7 plt.ylabel("Pagos en euros", fontsize = 12)
8 plt.plot()

```

## Código para comparativa de prestamos

```

1 Capital = float(input("Importe del Préstamo: "))
2 tasa = float(input("Tipo de interés del Préstamo (Ej. 8% = 0.08): "))
3 años = int(input("Periodo del Préstamo (años): "))
4 meses = int(input("Introduzca 0 para préstamos anuales o 1 para préstamos mensuales"))
5 tipo = input("Tipo de Préstamo (Francés, Americano, Italiano): ").lower()
6
7 frances = ['frances', 'f']
8 americano = ['americano', 'a']
9 italiano = ['italiano', 'i']
10
11 if tipo in frances:
12     PFrances(Capital, tasa, años, meses)
13
14 elif tipo in americano:
15     PAmericano(Capital, tasa, años, meses)
16
17 elif tipo in italiano:
18     PIItaliano(Capital, tasa, años, meses)
19
20

```

## 5.1.9 Beta de compañías

## Código para gráficos

```
1 data['Rentabilidad Valor'].cumsum().plot(subplots=True, figsize=(16,7))
```

```

1 def plot_regression(renta, asset):
2     plt.figure(figsize=(10,5))
3     sns.scatterplot('^IBEX', asset, data=renta,
4                     hue=renta.index.year, alpha=0.6, legend=False)
5     sns.regplot('^IBEX', asset, data=renta, scatter=False, color='green')
6
7 asset = 'AMS.MC'
8 plot_regression(renta, asset)

```

## Código para automatizar el cálculo de Beta

```

1 def beta(valor,periodo,mercado):      #asignamos las variables
2     simbol = [valor, mercado]          #creamos una lista para después extraer los datos de la página de yahoo
3     data = yf.download(simbol, period=periodo, auto_adjust=True) #descargamos los datos que necesitamos
4     renta = np.log(1 + data.loc[:, 'Close'].pct_change()).dropna(how = 'all') #calculamos la rentabilidad
5     var = renta[(mercado)].var() #calculamos la varianza del mercado
6     cov = renta.cov()           #calculamos la covarianza del valor
7     beta = renta.cov()/var      #calculamos la Beta de la compañía
8     return beta[mercado].head(1).to_frame('Beta').T #Finalmente mostramos el dato que necesitamos.
9

1 valor = 'IAG.MC' #recordemos que debemos indicar el símbolo de la empresa.
2 periodo = '5y'   #podemos seleccionar varios períodos. (1d, 5d, 1mo, 3mo, 6mo, 1y, 2y, 5y, 10y, ytd, max)
3 mercado = '^IBEX'
4
5 beta(valor, periodo, mercado)

[*****100%*****] 2 of 2 completed

IAG.MC
Beta 1.664964

```

### 5.1.10 Automatización de informes

Código para gráficos

```

1 plt.figure(figsize=(6,10))
2 sector.plot(kind='pie', subplots=True, fontsize=12, ylabel='', colormap = 'mako')
3 plt.title("Sectores mas influyentes en el IBEX35", fontsize=20)
4
5 plt.savefig("Ibex.jpg", bbox_inches='tight') #nos guarda la imagen
6
7 plt.show()

```

```

1 listaEmpresas = unido.groupby('Sector').Empresa.count().sort_values(ascending=False)
2 listaEmpresas.plot(kind='barh', figsize=(6,6), fontsize=13, color="darkblue")
3 plt.title("Número de empresas por Sectores en Ibex35")
4
5 plt.savefig("NumeroEmpresas.jpg", bbox_inches='tight')
6
7 plt.show()

```

```

1 lugar = unido.groupby('Sede').Empresa.count().nlargest(5)
2 lugar.plot(kind='pie', subplots=True, figsize=(6,10), fontsize=13, ylabel='', colormap='mako')
3 plt.title('Ciudades con más empresas del Ibex', fontsize=20)
4
5 plt.savefig("Ciudades.jpg", bbox_inches='tight') #es importante guardar la imagen antes de visualizarla,
6
7 plt.show()
8

```

```

1 entrada = unido.groupby('Entrada').Empresa.count()
2 entrada.plot(kind='bar', subplots=True, figsize=(9,3), fontsize=10, ylabel='', color='darkblue')
3 plt.title("Nº de Empresa que han entrado en el Ibex por año", fontsize=20)
4 plt.ylabel('Año Entrada')
5 plt.xlabel('Nº de Empresas')
6
7 plt.savefig("Empresa.jpg", bbox_inches='tight')
8
9
10 plt.show()
11

```

```

1 evoluci['^IBEX'].plot(figsize=(10,5));
2 plt.title('Evolución Ibex35 en los últimos meses', fontsize=15)
3 plt.ylabel('Precio de Cierre')
4 plt.xlabel('Fechas')
5 plt.yticks(range(7800,8700,100));
6
7 plt.savefig("Evolucion.jpg", bbox_inches='tight')
8
9 plt.show()

```

Código completo automatización de informes

```

document = Document()
document.add_heading("ANÁLISIS DE IBEX_35", level=0)
document.add_heading("1. Características del Ibex35", level=1)
document.add_paragraph("El Ibex se compone de las 35 empresas con más liquidez que cotizan en el sistema bursátil español que está formado por las bolsas de Madrid, Valencia, Barcelona y Bilbao. Por ello se utiliza como referencia para conocer la situación de la Bolsa española.")
document.add_paragraph("Si comprobamos la distribución por sectores, podemos comprobar que Electricidad y Gas es el sector más representativo, ya que supone un 22% del Ibex, seguido de la Banca, Textil y calzado, Telecomunicaciones y Minerales-transformación.")

document.add_picture("ibex.jpg", width=Cm(11))

document.add_paragraph("El gráfico nos muestra los 5 sectores más representativos, siendo un total de 17 sectores los que están representados en el Ibex.35")
document.add_paragraph("En la siguiente tabla comprobamos la distribución por sectores del Ibex")

table = document.add_table(rows=1, cols=2, style='Colorful Shading Accent 1')
#font = row_cells[0].text
table.height = Pt(8)

table.rows[0].cells[0].text = 'Sectores'

```

```



```

Informe automatizado (2 páginas)

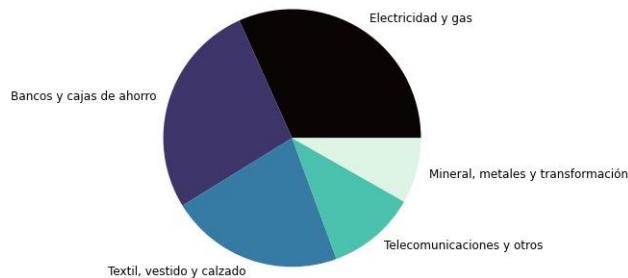
# ANÁLISIS DE IBEX\_35

## 6 1. Características del Ibex35

El Ibex se compone de las 35 empresas con más liquidez que cotizan en el sistema bursátil español que está formado por las bolsas de Madrid, Valencia, Barcelona y Bilbao. Por ello se utiliza como referencia para conocer la situación de la Bolsa española.

Si comprobamos la distribución por sectores, podemos comprobar que Electricidad y Gas es el sector más representativo, ya que supone un 22% del Ibex, seguido de la Banca, Textil y calzado, Telecomunicaciones y Minerales-transformación.

Sectores mas influyentes en el IBEX35

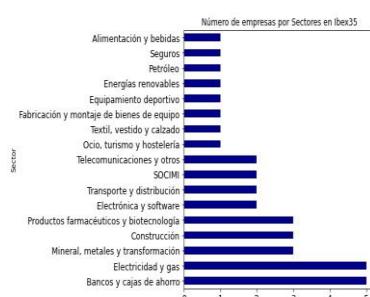


El gráfico nos muestra los 5 sectores más representativos, siendo un total de 17 sectores los que están representados en el Ibex.35

En la siguiente tabla comprobamos la distribución por sectores del Ibex

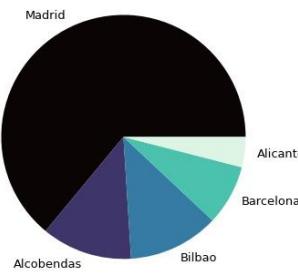
Sectores	Ponderación
Electricidad y gas	22.19%
Bancos y cajas de ahorro	18.95%
Textil, vestido y calzado	15.25%
Telecomunicaciones y otros	7.83%
Mineral, metales y transformación	5.75%
Construcción	5.74%
Transporte y distribución	5.63%
Electrónica y software	4.97%
Fabricación y montaje de bienes de equipo	3.91%
Petróleo	2.99%
Productos farmacéuticos y biotecnología	2.38%
SOCIMI	1.44%
Seguros	0.95%
Equipamiento deportivo	0.83%
Alimentación y bebidas	0.48%
Energías renovables	0.39%
Ocio, turismo y hostelería	0.24%

En el siguiente gráfico podemos comprobar el número de empresas totales por sector



En el siguiente gráfico vemos las 5 ciudades donde más empresas tienen sus sedes.

Ciudades con más empresas del Ibex

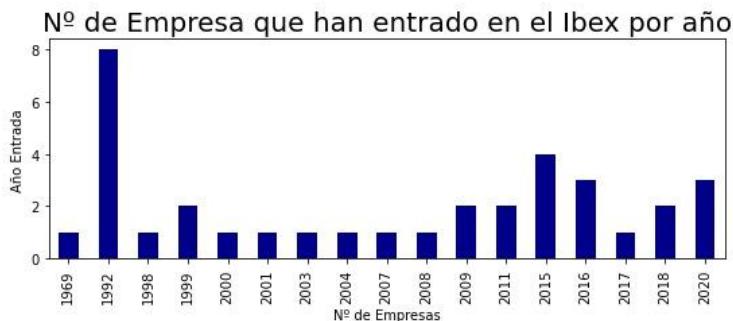


Madrid es la ciudad donde más empresas tienen su sede, seguido de Alcobendas y Bilbao. Después le siguen Barcelona y Alicante.

Possiblemente podemos encontrar distintos motivos por los que las empresas deciden establecer su sede en un punto geográfico y no otro, así las infraestructuras, la innovación tecnológica o su capital humano son factores que influyen a la hora de establecer la sede de una empresa.

El Ibex se creó en 1.992, por ello no es de extrañar que nos encontremos con el año en el que más entradas se produjeron en el Ibex.

En el siguiente gráfico podemos ver el número de entradas por año.



Podemos ver una entrada en el año 1969, que probablemente sea un error publicado en la fuente de origen de los datos, Wikipedia, puesto que la creación del Ibex fue en 1.992, y por tanto el año 1969 debe ser el año de creación de la propia empresa.

En cualquier caso, el código completo se encuentra en el repositorio de GitHub donde se puede descargar de forma libre.

## 7 Referencias

Aparicio, Adolfo. *Alto Código*. s.f. <https://altocodigo.blogspot.com/2018/06/1-hola-mundo-en-python.html> (último acceso: 2020).

Laca, Mariano. <https://pythones.net/clases-y-metodos-python-oop/>. s.f.

## 8 Bibliografía

- Aparicio, Adolfo. *Alto Código*. s.f. <https://altocodigo.blogspot.com/2018/06/1-hola-mundo-en-python.html> (último acceso: 2020).
- . *www.masterfinanciero.es*. s.f. <https://www.masterfinanciero.es/2011/09/clasificacion-de-las-rentas.html> (último acceso: 27 de 05 de 2021).
- Dedov, Florian. *Python Bible for Finance*. Florian Dedov, 2019.
- Díaz, Juan. *Pildoras informáticas*. s.f. <https://www.youtube.com/playlist?list=PLU8oAIHdN5BlvPxziopYZRd55pdqFwkeS> (último acceso: 2021).
- Hagmann, Alexander. *Curso Python de Udemy*. 01 de 02 de 2021. <https://www.udemy.com/home/my-courses/learning/> (último acceso: 2021).
- Hilpisch, Yves. *Python for Finance*. O'Reilly Media, 2019.
- Ironhack. *Webminar Python*. s.f. [https://www.crowdcast.io/e/python-office/register?utm\\_source=profile&utm\\_medium=profile\\_web&utm\\_campaign=profile](https://www.crowdcast.io/e/python-office/register?utm_source=profile&utm_medium=profile_web&utm_campaign=profile).
- Laca, Mariano. <https://pythones.net/clases-y-metodos-python-oop/>. s.f.
- Pla-Santamaría, Francisco Salas-Molina y David. *Gestión de Tesorería con Python*. Universitat Politècnica de Valencia, 2017.
- Python, Comunidad usuarios. *Pybonacci*. s.f. <https://pybonacci.org/tag/openpyxl/>.
- Yan, Yuxing. *Python for Finance*. Canada: Yves Hilpisch, 2019.