**PRACTICAS DE PYTHON:**

1. **Control de flujo y colecciones**: Se piden dos números, uno menor que 10 y otro mayor que 100. El primer número representa el número de intervalos y hay que calcular los rangos de los intervalos según el segundo número. Por ejemplo: n1 = 4, n2 = 100. Los rangos serían de 0..24, 25..49, 50..74, 75..99.

Después se generan número al azar que no sobrepasen el número n2 y se almacenan en cada intervalo. Luego listar los intervalos con los números que se han recogido.

1. **Funciones**: Comprobar si todos los elementos de una lista son iguales a un número dado. Implementarlo de forma recursiva.
2. **Objetos**: Define una clase **CoordenadaGeo** y las que sean necesarias para que se puedan hacer operaciones con ella, como por ejemplo: calcular las antípodas, a partir de dos coordenadas dadas que define una región (con la esquina sup. Izq e inf. Derecha, comprobar si la coordenada se encuentra dentro o fuera). Imprimirlas en formato: 40º 24’ N, 3º 41’ 15” W

Las coordenadas se crearán así: con dos números reales.

Si Latitud es: 40.24 🡪 indica latitud Norte, si < 0 será Sur.

Si Longitud es: -3.6875 🡪 indica longitud Oeste, si es > 0, será Este.

madrid = coordenadaGeo(40.4, -3.6875)

print('Madrid', madrid)



1. **Acceso a Web**:

A partir de esta URL:

http://maps.googleapis.com/maps/api/geocode/json?address=**madrid**&sensor=false&language=es

Google nos devuelve una estructura en formato JSON, se trata de acceder a esta dirección con la ciudad que queramos, ver lo que devuelve y después convertirlo a formato json, se puede utilizar:

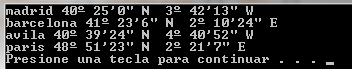
***import json***

***dic = json.loads(contenido\_leido\_de\_la\_URL)***

*Nos devuelve un diccionario anidado con listas, nos interesa el campo* ***location*** *con* ***lat*** *y* ***lng****.*

*Codificar el programa en una clase y un método que reciba la ciudad y nos devuelva las coordenadas de la ciudad. Utilizar las clases anteriores.*

***# Tener en cuenta que la cadena que nos devuelve viene en formato bytes, la cadena se precede de una b, utilizar la función decode(‘utf-8’) para pasarlo a Unicode.***

**

1. **Objetos**: Implementar una jerarquía de clases que nos permita representar figuras en 2D y en 3D, las operaciones que queremos tener son sencillas. Para 2D cálculo de áreas y para 3D cálculo de volúmenes. Todas las clases dispondrán de un método visualizar() que mostrará los datos de cada figura. Trabajar con Circulo, Cuadrado, Triangulo, Cubo y Cilindro. Trabajar a distintos niveles definiendo listas de Figuras, de Figuras2D y Figuras3D.
2. **Polimorfismo**: Se trata de implementar una clase Personal que gestiona todos los empleados de una empresa. Dentro de la empresa tenemos distintos perfiles: Director, Administrativo y Jefe de Proyecto. En común se almacena el nombre, apellidos, código de empresa y sueldo. A parte cada perfil añade más información, en el caso del Administrativo dispone de dos pagas extra. El jefe de proyecto, tiene su sueldo base y una parte variable: incentivos. El director tiene el sueldo base, una paga de beneficios y unos objetivos.

El cálculo del nuevo sueldo se hace según una tabla de baremos. Por ejemplo:

* + **IPC: %**
  + **extras: ± %**
  + **objetivos: ± importe en €**
  + **incentivos: ± importe en €**

. ¿Qué relaciones hay entre las clases y que tipo de relación?

**Desde el módulo main podríamos hacer algo así:**

**P = Personal ()**

**p.añadir(unEmpleado)**

**p.darDeBaja(codigo) // No se elimina físicamente se pone una marca. Devuelve true si lo ha encontrado.**

**p.listar() // Muestra la información de TODOS los empleados con todos sus datos. Podemos indicar los que están de baja con un \*.**

**p.subirSueldo(Tabla\_baremos)**