Pruebas de selección IoT Team

Para las siguientes pruebas adicionalmente a su objetivo se valorará:

- La pulcritud del código.
- El diseño del código.
- La brevedad del código

Programación Python:

Prueba 1: Recolección y tratamiento:

Implementar la función **get_api_data :**

- **Entrada:** String
- **Salida:** List<Tuple<key, value>>
- **Definición:**
  La función debe recoger el json de tipo <key, value> de la url pasada como entrada, y devolver-lo como una lista de tuplas <key,value>.
- **Url test:**
  - https://invelonjobinterview.herokuapp.com/api/test1
  - https://invelonjobinterview.herokuapp.com/api/test2

Ejemplos:	
Json recibido	**Salida esperada**

<pre>{
"name": "Raul",
"email": "test@test.com",
"preferences": [1, 3, 7],
"affiliate": true
}</pre>
	<pre>(
	["name", "Raul"],
	["email", "test@test.com"],
	["preferences", [1, 3, 7]],
	["affiliate", True]
	)</pre>

<pre>{
"users": [
{
"name": "Raul",
"email": "test@test.com",
"preferences": [1, 3, 7],
"affiliate": true
},</pre>
	<pre>('users',[
	{
	'name': 'Raul',
	'email': 'test@test.com',
	'preferences': [1, 3, 7],
	'affiliate': True
	},
	{</pre>

```
    {
      "name": "Marino",
      "email": "test2@test.com",
      "preferences": [2, 5],
      "affiliate": false
    }
  ]
}
```

```
        'name': 'Marino',
        'email': 'test2@test.com',
        'preferences': [2, 5],
        'affiliate': False
    }
])
```

Prueba 2: Concurrencia:

Implementar la función **concurrency_with _coffee :**

- **Entrada:** Sin parámetros de entrada
- **Salida:** None
- **Descripción:**
  La función debe mostrar por pantalla la frase **“- I'm tired, Bob!”** y entonces crear 5 threads, estos mostraran palabras sin realizar saltos de línea.
  Finalmente el programa esperará la finalización de estos thread e imprimirá la frase **“- You’re right!”**

  Frases de los threads:
  - **Thread 1: Mostrar:** “are made of coffee? ”
  - **Thread 2: Mostrar:** “Do you know ”
  - **Thread 3: Mostrar:** “it!”
  - **Thread 4: Mostrar:** “ that our bodies”
  - **Thread 5: Mostrar:** “-Try drinking ”

  Importante: Los threads se deben iniciar en orden (1, 2, 3, 4, 5) pero su orden de ejecución (cuando muestran su frase por pantalla) debe ser 2, 4, 1, 5, 3. El tiempo de ejecución del programa debe ser lo más óptimo posible.

Programación Bash:

Prueba 1: Creación de ficheros con parámetros:

Crear un script bash con el nombre **i_am_the_master_of_files:**

Especificación: i_am_the_master_of_files <file_name> <uuid>

El script creará un archivo en el directorio con el nombre especificado en la variable **filename** y le quitara los derechos de lectura públicos. (Para el propietario o el grupo del propietario se mantendrán los permisos por defecto igual).
El contenido del archivo será el siguiente, substituyendo {{uuid}} por el valor que se especifique en la variable de entrada uuid:

```
[FILE]
folder = ./uploads
id = {{uuid}}
slots = 2
```

Prueba 2: Modificación de ficheros con parámetros:

Crear un script de bash con el nombre **i_can_alter_your_reality**

Especificación: i_can_alter_your_reality <file_name> <uuid>

El script accederá al archivo con el nombre especificado en la variable **filename** buscará los campos con identificador **id** y sustituirá su valor por el de la variable **uuid**
Consideración: El archivo siempre tendrá el mismo formato (pero con un número no determinado de variables):

Formato

```
[TITULO]
variable1 = Texto
variable2 = texto
id = 21938210ass231323
variable3 = Texto
```

Ejemplo:

```
./i_can_alter_your_reality myfile.cfg 33
```

myfile.cfg (Original)	**myfile.cfg (Alterado)**

`[FILE]`
`folder = ./uploads`
 	
`id = 55`
 	
`max_users = 3`
`max_slots = 2`
	`[FILE]`
	`folder = ./uploads`
	
	`id = 33`
	
	`max_users = 3`
	`max_slots = 2`

Conocimiento de Django Backend:

Prueba 1: Creación aplicación básica:

Utilizando el Framework Web django crea un proyecto llamado **CoffeeControl** y añadir 2 apps: **logistics y business.**

Importante: Estas apps han de estar correctamente instaladas en el proyecto.

Prueba 2: Configuración de las entidades de la base de datos:

En esta prueba se valorará cada punto de cada apartado de forma individual.

Parte 2.1: Setup mínimo

Partiendo de "Prueba 1: Creación aplicación básica" se debe configurar las siguientes entidades en cada app:

- **bussines:**
  - **Department:**
    - **name:** String
  - **Worker**:
    - **user**: Relacion con User de django. User 1 -> 1 Worker
    - **department:** Relación con Department. Department 1 -> * Workers

- **logistics:**
    - **CoffeType**
        - **name:** String
        - **current_price:** Float (No puede ser inferior a 0)
    - **CoffeeStock:**
        - **current_units:** Integer (No puede ser inferior a 0)
        - **coffee_type:** Relación con CoffeeType. CoffeType 1 -> * CoffeeStock
        - **department:** Relación con bussines.Department. Department 1 -> * CoffeeStock
    - **CoffeeOrder**:
        - **unit_price**: Float (No puede ser inferior a 0)
        - **units**: Integer (No puede ser inferior a 0)
        - **status**: String (Opciones: Accepted | Pending.) Default: Pending
        - **coffee_stock**: Relación cón CoffeStock. CoffeeStock 1 -> * CoffeeOrder

Parte 2.2: Setup Avanzado:

Adicionalmente al apartado "Parte 2.1: Setup mínimo", añade las siguientes condiciones:

1. Añadir la restricción de que solo puede existir un CoffeStock por Department y CoffeType.
2. Al crear una entrada a CoffeType automáticamente se ha de crear una entrada a CoffeStock por cada uno de los registros en Department, con valor 0 en current_units.
3. Al crear una entrada a Department automáticamente se ha de crear una entrada a CoffeStock por cada uno de los registros en CoffeType, con valor 0 en current_unit.
4. Cuando el valor de un CoffeStock sea 0 (ya sea por una modificación o por que se acaba de crear), automáticamente se creara un registro en CoffeOrder teniendo en cuenta:
    a. **unit_price**: El **current_price** del tipo de café del stock que se está pidiendo.
    b. **units**: 100
    c. **status**: "pending"