Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-2

13 de Diciembre de 2021
Actividad Bonus

Actividad Bonus

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AB/`
- **Hora del *push*:** Viernes 17 de diciembre a las 20:00

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción

Se acerca el verano y empiezas a prepararte para las vacaciones. El semestre ha sido duro y no has tenido tiempo para jugar tu videojuego favorito: Pokémon. Es por esto, que decides retomarlo después de tanto tiempo.

Sin embargo, debido a problemas desconocidos, ¡te das cuenta que has perdido todo lo avanzado! Por suerte, se hizo una copia de seguridad de ciertos datos necesarios para recuperar tu avance. ¡Vas a tener que utilizar tus conocimientos sobre *Web Services* para poder recuperarlos a través de *requests*!

Flujo del Programa

Tienes a disposición dos APIs sobre las cuales deberás hacer *requests*. La primera es una API perteneciente al curso, con la cual deberás obtener parte de la información necesaria para actividad y con la cual podrás ir probando tu avance. La segunda es la API oficial de tu juego de Pokémon, donde se guarda información específica sobre ellos y sus distintas características.

El juego almacena habilidades adquiridas por los pokemones utilizados por el usuario. La **habilidad** especial de tu usuario es algo que se ha perdido. Es por esto, que deberás, primero recuperarla, para luego obtener los **pokemones** que tengan esa habilidad. Cada pokémon tiene ciertos atributos de interés que deberás obtener para, finalmente, calcular ciertas **estadísticas** en base a ellos. Los datos de las estadísticas servirán para recuperar el avance perdido del juego.

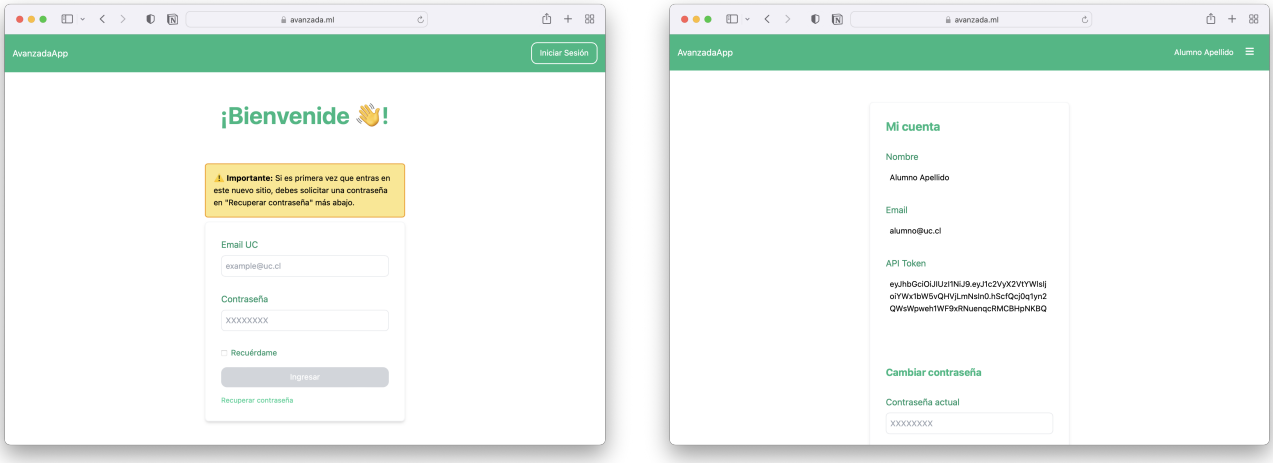
Archivos

- `main.py`: Es el archivo principal del programa. Se encarga de llamar las funciones necesarias para recuperar los datos.
- `api_curso.py`: Se encarga de recuperar la habilidad de tu usuario utilizando la API dispuesta por el equipo docente.
- `pokemon.py`: Se encarga de recuperar la información sobre los pokemones y generar las estadísticas utilizando la API oficial de tu videojuego.

Parte I: Obtener Habilidad

Gracias a la API dispuesta por el equipo docente IIC2233, puedes recuperar tu habilidad especial que usabas en el juego. Debes hacer el `request` tipo `GET` al sitio `https://www.avanzada.ml/api/v2/bonus/ability`, el cual te asignará y entregará una habilidad de pokémon. Una vez que obtengas la información de la habilidad, podrás desbloquear la siguiente etapa!

Para acceder a la API del curso necesitas identificarte mediante un API Token[1] que es lo mismo que el API Key visto en los contenidos del curso, el cual puedes obtener a través de este **sitio web**. Para obtener el Token, debes acceder a tu cuenta, con tu email UC y la contraseña debes solicitarla según se indica en la página (ver Figura 1a).



(a) Formulario de inicio de sesión.

(b) Vista de la página "Mi cuenta".

Figura 1: Vistas del sitio web `www.avanzada.ml`.

Una vez que ingresas, debes dirigirte al Menú "Mi cuenta"donde encontrarás tu API Token personal como se observa en la Figura 1b, el cual es intransferible. [2]

Con este Token puedes hacer un *request* al sitio de la siguiente forma:

```
GET https://www.avanzada.ml/api/v2/bonus/ability?api_token=PERSONAL_TOKEN
```

Donde `PERSONAL_TOKEN` corresponde al Token obtenido anteriormente. La respuesta a una consulta correcta será lo siguiente:

[1] Un API Token es un token único personal que permite que la API pueda identificar quién está realizando una *request*.
[2] Este token no cambia ni tiene expiración. Frente a cualquier problema contactar al equipo docente.

```
{
    "ability": {
        "name": "nombre_habilidad",
        "url": "https://pokeapi.co/api/v2/ability/id_habilidad"
    },
    "student": {
        "id": "21328847",
        "fullname": "Alumno Apellido",
        "section": 5,
        "email": "alumno@uc.cl"
    }
}
```

Para implementar este llamado a la API, debes completar la función `info_api_curso` en el archivo `api_curso.py`.

`def info_api_curso(token: str) -> dict`: Esta función recibe el API Token personal que obtuviste desde el sitio web y debe retornar la respuesta de la API en formato diccionario si es que la consulta fue exitosa o un diccionario vacío en otro caso.

Parte II: Obtener Pokemones

En esta parte, deberás modificar el archivo `pokemon.py` y utilizar la API del juego de Pokémon. La documentación se encuentra en **este link**.

Esta parte consiste en dos *requests*. El primero es para obtener información de la habilidad, la cual incluye una lista de pokemones asociados a esa habilidad. El segundo es para obtener información sobre cada uno de los pokemones.

Primero, deberás completar la siguiente función:

`def obtener_info_habilidad(url: str) -> dict`: Recibe como parámetro el `url` de tu habilidad y deberás hacer un `request` al sitio descrito en `https://pokeapi.co/docs/v2#abilities` para obtener la información asociada a esa habilidad. Esto te entregará un mensaje en formato JSON que deberás parsear para generar y retornar un `dict` con los siguientes datos:

- `"name"`: deberás guardar el nombre de la habilidad obtenida en la respuesta en formato `str`.
- `"effect_entries"`: deberás guardar la descripción del efecto de la habilidad en formato `str` solo para el idioma inglés (`"name": "en"`). Para esto, deberás obtener el `str` almacenado en `short_effect`.
- `"pokemon"`: Deberás crear una lista de pokemones. Cada pokémon debe ser un diccionario que contenga una llave `"name"` que representa el nombre del pokémon en `str`, y una llave `"url"` con la dirección `url` específica del pokémon, también en `str`.

Una vez que hayas obtenido la información solicitada, podrás completar la siguiente función:

`def obtener_info_pokemones(pokemones: list) -> list`: Recibe una lista de pokemones como la generada en la función anterior y para cada uno de ellos debes hacer un *request* a la `url` asociada a cada pokémon, la cual se describe en `https://pokeapi.co/docs/v2#pokemon`.

Con la respuesta de cada *request*, deberás generar y retornar una lista de diccionarios que contengan los siguientes datos para cada pokémon:

- `"id"`: Representa la identificación del pokémon y deberás almacenarlo como `int`.

- `"name"`: Representa el nombre del pokémon y debe ser un `str`.
- `"height"`: Representa la altura del pokémon y deberás almacenarlo como `int`.
- `"weight"`: Representa el peso del pokémon y deberás almacenarlo como `int`.
- `"stats"`: Debe ser un diccionario con las estadísticas del pokémon almacenada en el objeto `"stats"` de la repuesta de la API, pero deberás convertirlo al siguiente formato:

```
{
    "nombre_stat": {
        "base_stat": int,
        "effort": int
    },
    ...
}
```

- `"types"` Es una lista que contiene los nombres (`str`) de los tipos de pokémon al cual el pokémon pertenece. Los tipos de cada pokémon se encuentran en el objeto `"types"` de la respuesta de la API.

Parte III: Generar estadísticas

¡Ahora, podrás finalmente recuperar tu avance! Para esto, deberás generar algunas estadísticas a partir de la información obtenida. Estas son:

- `def obtener_pokemon_mas_alto(pokemones: list) -> str`: A partir de la lista de pokemones, deberás obtener el pokémon tal que su `"height"` sea el valor más alto. Deberás retornar el nombre de ese pokémon en formato `str`.
- `def obtener_pokemon_mas_rapido(pokemones: list) -> str`: A partir de la lista de pokemones, deberás obtener el pokémon tal que su velocidad sea la más alta. Para esto, deberás usar la llave `"stats"` y verificar si contiene `"speed"` como nombre. En caso de tenerlo, el valor de la velocidad para ese pokémon se encontrará en la llave `"base_stat"`. En caso contrario, puedes asumir que su velocidad será cero. Deberás retornar el nombre del pokémon con velocidad más alta en formato `str`.
- `def obtener_mejores_atacantes(pokemones: list) -> list`: A partir de la lista de pokemones, deberás entregar una lista con los los 5 mejores pokémon atacantes ordenados de forma descendente. Para calcular que tan buen atacante es un pokémon debes calcular:

  $$ATK/DEF$$

  Donde ATK es el valor `"base_stat"` del atributo `"attack"` y DEF es el valor `"base_stat"` del atributo `"defense"`, obtenidos a partir del diccionario `"stats"` del pokemon.

  En caso de un pokemon no posea uno de los atributos (`"attack"` o `"defense"`), debe ser ignorado. Si la lista resultante posee menos de 5 pokemones, entonces debes retornar toda la lista manteniendo el orden descendente.
- `def obtener_pokemones_por_tipo(pokemones: list) -> dict` A partir de la lista de pokemones, deberás generar y retornar un diccionario, donde la llave será el nombre de un tipo de pokémon y el valor una lista con los nombres de los pokemones que pertenecen a ese tipo.

  Por ejemplo, si tienes el pokémon `Zapdos` que es de tipo `"electric"` y `"flying"`, y tienes al pokémon `Pikachu` que es de tipo `"electric"`, entonces el resultado a esta función debería ser algo así:

```
    {
        "electric": ["zapdos", "pikachu"],
        "flying": ["zapdos"]
    }
```

Una vez que hayas calculado la información solicitada, habrás finalmente recuperado el avance en el juego, y podrás tener unas merecidas vacaciones.

Parte IV: Test con API

Durante tu progreso en la actividad, tendrás la posibilidad de auto evaluar tu avance al mismo tiempo que aplicas los contenidos. Para ello dispondrás de una `url` en la API del curso, en donde podrás probar tus resultados.

La API del curso dispone de la `url` `https://www.avanzada.ml/api/v2/bonus/tests/:id` la cual recibe consultas del tipo `POST` para ejecutar el test indicado en el parámetro `:id`. Para utilizar esta `url` es necesario además el API Token personal. La información a enviar debe ser un diccionario que contenga la llave `"test"`, el cual será otro diccionario con los siguientes atributos:

- `"function_name"`: Corresponde al nombre de la función a testear, debe ser el nombre correspondiente al id que se está consultando.
- `"function_response"`: Corresponde a la información retornada por la función, ya sea un diccionario, una lista o un string.

Un ejemplo de consulta sería:

```
POST https://www.avanzada.ml/api/v2/bonus/tests/5?api_token=PERSONAL_TOKEN
DATA:
```

```
{
    "test": {
        "function_name": "obtener_mejores_atacantes",
        "function_response": [
            { "name": "sandile", "attack": 72, "defense": 35 },
            { "name": "shinx", "attack": 65, "defense": 34 },
            { "name": "krokorok", "attack": 82, "defense": 45 },
            { "name": "luxio", "attack": 85, "defense": 49 },
            { "name": "staraptor", "attack": 120, "defense": 70 }
        ]
    }
}
```

Lo cual para una consulta correcta entregaría la siguiente respuesta en caso de que el test haya resultado exitoso:

```
{
    "result": "success"
}
```

Pero para una consulta correcta donde el test no sea existoso se obtendría una respuesta como la siguiente:

```
{
    "result": "error",
```

```
    "message": "El formato no coincide con el formato solicitado"
}
```

Los `ids` para las diferentes funciones implementadas se detalla en la siguiente tabla:

Id	Función	Resultado
1	`obtener_info_habilidad()`	`dict`
2	`obtener_info_pokemon()`	`list`
3	`obtener_pokemon_mas_alto()`	`str`
4	`obtener_pokemon_mas_rapido()`	`str`
5	`obtener_mejores_atacantes()`	`list`
6	`obtener_pokemones_por_tipo()`	`dict`

Para poder utilizar esta funcionalidad deberás completar la siguiente función en el archivo `api_curso.py`.

`def enviar_test(token: str, test_id: int, respuesta: Object) -> bool`: Esta función recibe como argumentos, el API Token que obtienes desde el sitio web, un entero correspondiente al `id` del test a ejecutar y un argumento respuesta que contiene la respuesta de la función a testear.

Debes hacer un `request` del tipo `POST` al sitio mencionado más arriba, donde el `:id` es el `test_id` obtenido como argumento. También debes generar el diccionario con la información a enviar a la API, según el formato indicado más arriba, incluyendo el argumento `respuesta`.

La función debe retornar un `bool` dependiendo de si el test fue exitoso o no. En caso de de que haya ocurrido un error en la consulta, se debe retornar `False`.

Consideraciones importantes de entrega

La corrección de esta evaluación será automática. Esto quiere decir que el puntaje asignado depende directamente de si las funcionalidades están implementadas correctamente o no, y no pasará por la corrección detallada del equipo de ayudantes. Para lograr esto, se utilizarán archivos de *test* que ejecutarán tu código, y a partir de este, obtener un *output* definido. Si este *output* coincide con el esperado, se considerará ese *test* como correcto.

Para cada función, se realizarán múltiples y variados *tests*, de distinta complejidad, que buscarán detectar la correctitud de la implementación. Cada *test* se considerará correcto solo si no ocurre una excepción durante la ejecución de la función y si el resultado coincide con el esperado. Si todos los *test* de una función son correctos, se asignará puntaje completo asignado a la función. De la misma forma, se asignará la mitad del puntaje si al menos el 75 % de los *test* son correctos. En caso contrario, no se asignará puntaje a la implementación de esa función.

En esta evaluación **no** se recibirán entregas atrasadas. Cualquier *commit* ***pusheado*** posterior a la hora de entrega no se considerará. Se aconseja, como siempre, **realizar *commits* y *pushs* parciales de sus avances**. Por ejemplo, al terminar de implementar una función, es un buen momento para subir un avance, de manera que no acumulen muchos cambios nuevos al hacer *push* cerca de la hora de entrega.

Para esta evaluación **no se evaluará el uso de `.gitignore`**. De todas formas, lo único necesario que debes entregar son los **módulos a completar**: `api_curso.py` y `pokemon.py`. No pasa nada si se sube `main.py`.

Tampoco habrá entrega de `README`, ya que la corrección será automática. Recuerda seguir la estructura de archivos entregada y **no alterar los nombres de las funciones a completar**, para asegurar estas puedan ser correctamente importadas.

Como mencionado al comienzo de este enunciado, se espera trabajes y entregues en tu repositorio personal, y en una carpeta **nueva** llamada `AB/` dentro de la carptea `Actividades`. Se espera que usen los módulos base siguiendo la misma estructura que en el código entregado, al primer nivel de la carpeta `AB/`:

```
Actividades/
└── AB/
    ├── main.py
    ├── api_curso.py
    └── pokemon.py
```

Notas

- Te recomendamos leer con calma y detenidamente la documentación de los sitios mencionados. Te ayudará a comprender bien la información que recibirás como respuesta.
- Recuerda que siempre puedes revisar las respuesta de las consultas que estás haciendo imprimiéndolas en consola.
- Para correr el programa, puedes ejecutar el archivo `main.py` y también puedes comentar las líneas de las funciones que aún no has implementado.

Requerimientos

- (1.00 pts) Parte I: Obtener Habilidad
- (2.00 pts) Parte II: Obtener Pokemones
- (2.00 pts) Parte III: Generar estadísticas
- (1.00 pts) Parte IV: Test con API