Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2233 Programación Avanzada (2018-2)

Tarea 05

Entrega

- **Tarea**
  - **Fecha y hora:** domingo 2 de diciembre de 2018, 23:59
  - **Lugar:** GitHub — Carpeta: `Tareas/T05/`
- `README.md`
  - **Fecha y hora:** domingo 2 de diciembre de 2018, 23:59 (ojo, «domingo» es distinto de «lunes»)
  - **Lugar:** GitHub — Carpeta: `Tareas/T05/`

Objetivos

- Utilizar distintos *web services* para obtener información de un contexto en particular.
- Buscar, estudiar y utilizar la documentación de diferentes API.
- Aplicar expresiones regulares para resolver un problema.
- Sintetizar conocimientos estudiados a lo largo del curso.
- Y probablemente, salvar el ramo.

Índice

1. Introducción a DCConnect

Luego del exitoso desarrollo de DCCurve, se presenta el último desafío del semestre. Para esta ocasión *Tini* necesita desplegar a sus secuaces por todo Santiago. El objetivo es tener *ojos* en todos los rincones de esta ciudad para controlar perfectamente su territorio. Tu misión es ayudar a *Tini* a través del desarrollo de **DCConnect** para que así pueda asistir al famoso «ombligo» de la G2016 y pueda carretear como corresponde luego de terminar este fabuloso ramo con estilo. Esta increíble aplicación cuenta con usos avanzados de distintas API ya implementadas y validaciones con expresiones regulares.

2. Flujo del programa

Esta sección describe brevemente cuál es el flujo esperado que tu programa debe tener.

2.1. Inicio de sesión

Para poder hacer uso de DCConnect es necesario acceder con una dirección de correo y una contraseña, las cuales deben ser validadas según lo descrito en la sección 4. No es necesario que guardes los usuarios en una base de datos; sólo debes validarlos.

2.2. Flujo de la información

DCConnect debe conectarse a tres API para realizar múltiples funciones. Estos son los pasos que tu programa debe seguir.

1. Iniciar sesión en la aplicación.
2. Encontrar un lugar al que se desea ir. Para esto, se debe permitir al usuario buscar por categorías o descripción y que luego pueda elegir alguno dentro de una lista entregada por el programa. Por cada uno de estos lugares se debe mostrar la información relacionada. Estos pueden ser comentarios, ubicación, horario u otra información relevante al contexto que entregue la API de **Foursquare**.
3. Buscar el paradero más cercano a la ubicación del lugar al que se quiere llegar. La API de Foursquare te entregará las coordenadas de latitud y longitud del lugar.
4. Buscar todos los microbuses que se detengan en ese paradero que se pretende utilizar.
5. Dados los microbuses que pasan por ese paradero, obtener todos los paraderos de aquellos buses.
6. Obtener la posición actual del usuario mediante la API de **ipstack**.
7. Dados los paraderos de los recorridos de las micros, obtener con la API de **Transantiago**, las coordenadas de cada uno de esos paraderos y ver si se está a un radio de 10 kilómetros desde donde uno se encuentra.
8. De haber un lugar que cumpla con la condición anterior, **mostrar** cuánto tiempo le falta a la micro para llegar al paradero más cercano al usuario, la distancia al lugar deseado y la duración estimada de este viaje. El formato acerca de cómo mostrar esta información queda a tu criterio, pero debe ser fácil de entender.

Estos pasos se explicarán con mayor profundidad en sus respectivas secciones. No obstante, para realizar todo lo pedido, debes conocer cómo funcionan las API (*e.g.* saber qué tipo de información te pueden dar

y entender cómo acceder a esta información). En las siguientes secciones se muestran algunos ejemplos de esto, pero es altamente recomendable examinar los comandos para cada una de ellas en sus respectivas documentaciones.

3. *Web services*

Esta sección describe los servicios *web* que deberás estudiar y utilizar para obtener la información que necesitará tu aplicación. Es importante mencionar que en esta tarea **no está permitido** utilizar librerías —conocidas como ***wrapper libraries***— externas que encapsulen el comportamiento de las API que serán descritas en las siguientes subsecciones. Por esto mismo, el objetivo es que tú mismo hagas las solicitudes (utilizando la librería **requests**, por ejemplo) que consultarán directamente a las API. En otras palabras, tú debes escribir las URL correctas para conseguir la información que alimentará a DCConnect.

3.1. Foursquare

Foursquare es un servicio que entrega información acerca de locales de comida o entretenimiento de acuerdo a una georreferencia o una localidad. Para hacer uso de esta API, es necesario tener una cuenta en **foursquare.com** y luego entrar a la sección de *developers*, desde donde se debe crear una *app*. Con esta, podrás recibir las claves de autenticación para realizar las solicitudes. Sólo es necesario que la *app* realice *requests* de georreferencia.

Nota: En un momento, el sitio te sugerirá subir el "nivel de la cuenta"pidiendo datos de una tarjeta de crédito. En esta etapa ya puedes cerrar la pestaña e ignorar la «sugerencia». Vuelve a tu página de inicio de *developer* y allí encontrarás tus credenciales de acceso.

La documentación para la API se encuentra **aquí** y puedes encontrar un ejemplo de uso si vas a la barra lateral bajo el ítem **Places API** – **Getting Started** y luego haces clic en la pestaña «**Make a Request**». Esto te permitirá familiarizarte con toda la información que la API provee.

Un ejemplo de consulta sería el siguiente:

```
url = 'https://api.foursquare.com/v2/venues/search'
params = dict(
  client_id='CLIENT_ID',
  client_secret='CLIENT_SECRET',
  intent='browse',
  v='20180323',
  ll='-33.4389493,-70.6323447',
  radius=100,
  limit=10
)
resp = requests.get(url=url, params=params)
data = resp.json()
```

El atributo `params` (como ya te imaginas) corresponde a los parámetros de la consulta y se encuentran explicados en la documentación. A continuación, te entregamos un resumen de los parámetros que puede llevar tu consulta.

- `intent` (requerido)
  Corresponde a una descripción general de la intención de la consulta. De esto depende qué resultados

se entregarán y con qué exactitud. No todos son compatibles, por lo que se recomienda sólo utilizar `'browse'`.

- `v` (requerido)
  Corresponde a la versión de la API. A pesar de que no está explicado claramente en la documentación, **debe** estar escrito tal como sale en el ejemplo.
- `ll` (requerido)
  Corresponde a las coordenadas geográficas de donde se quiere realizar la consulta.
- `radius` (requerido)
  Corresponde al radio de búsqueda **en metros** desde el punto indicado en `ll`.
- `query` (opcional)
  Aplica un filtro al resultado. Por ejemplo, `query = 'coffee'` entrega resultados que contienen `'coffee'` en el nombre.
- `categoryId` (opcional)
  Filtra por tipo de lugar o evento. Por ejemplo, `categoryId = '4d4b7105d754a06376d81259'` entrega resultados de categoría *nightlife*. Este parámetro puede recibir más de un *id*, mientras estén separados por comas. Puedes encontrar más información sobre las categorías **aquí**.
- `limit` (opcional)
  Corresponde a la cantidad máxima de resultados a mostrar. Por defecto, este valor es 50.

Los parámetros de la consulta pueden variar. Para más información, puedes dirigirte a la página de documentación que se encuentra más atrás.

3.2. ipstack

3.2.1. Localización actual

Ahora que sabemos el lugar de destino, sólo debemos encontrar cómo llegar. Lo primero a saber es nuestra ubicación actual. Para esto podemos usar la API de ipstack, cuya documentación se puede encontrar **aquí**. En esta encontrarás cómo obtener una llave (*i.e. access key*) para realizar consultas.

Por ejemplo, un resultado exitoso realizado a la ruta `http://api.ipstack.com/check?access_key="TU_KEY"`, debería verse así:


```
{
    "ip":"200.104.205.3",
    "type":"ipv4",
    "continent_code":"SA",
    "continent_name":"South America",
    "country_code":"CL",
    "country_name":"Chile",
    "region_code":"RM",
    "region_name":"Santiago Metropolitan",
    "city":"Santiago",
    "zip":null,
    "latitude":-33.45,
    "longitude":-70.6667,
    "location":
        {
```

```
            "geoname_id":3871336,
            "capital":"Santiago",
            "languages":
                [
                    {
                        "code":"es",
                        "name":"Spanish",
                        "native":"Español"
                    }
                ],
            "country_flag":"http://assets.ipstack.com/flags/cl.svg",
            "country_flag_emoji":"CL",
            "country_flag_emoji_unicode":"U+1F1E8 U+1F1F1",
            "calling_code":"56",
            "is_eu":false
        }
}
```


A partir de este resultado, puedes obtener tu latitud y longitud actual[1].

3.3. Transantiago

La documentación puedes encontrarla **aquí**. Para el uso de esta API no es necesario que nos identifiquemos de alguna manera; simplemente podemos hacer los *requests* a las rutas indicadas en la documentación.

3.3.1. Paraderos

Teniendo nuestra longitud y latitud actual, podemos utilizar la API de Transantiago para obtener el paradero más cercano. En particular, deberás estudiar **esta sección** de la documentación de la API, que explica cómo obtener la información de los paraderos cercanos, dada una longitud, latitud y un radio (opcional) de búsqueda. Esta consulta tiene la siguiente estructura:

```
/v1/stops?limit=<numero_resultados>&center_lon=<longitud>&center_lat=<latitud>
```

Por ejemplo, un resultado exitoso realizado a la ruta `https://api.scltrans.it/v1/stops?limit=2&center_lon=-33&center_lat=-70`, debería verse así:


```
{
    "has_next": true,
    "page_number": 1,
    "total_results": 7898,
    "total_pages": 7898,
    "results": [
            {
                "stop_lat": "-33.609836757",
                "stop_code": "PF994",
                "stop_lon": "-70.519213887",
                "agency_id": "TS",
```


[1]Para obtener tu posición, ipstack utiliza tu dirección IP. Esta podría ser referida a tu proveedor y, por consiguiente, no ser exacta. Puedes leer más información sobre esto en **este artículo**.

```
                "stop_id": "PF994",
                "stop_name": "PF994-11 De Septiembre / Esq. El Bambú"
            },
            {
                "stop_lat": "-33.608317229",
                "stop_code": "PF194",
                "stop_lon": "-70.517800000",
                "agency_id": "TS",
                "stop_id": "PF194",
                "stop_name": "PF194-El Volcán / Esq. Las Higueras"
            }
        ],
    "page_size": 2
}
```


3.3.2. Recorridos

Y ahora que ya sabemos cómo llegar al paradero, debemos averiguar cuánto demorará en llegar el siguiente bus. En particular, deberás estudiar **esta sección** de la documentación que explica cómo obtener la información de los buses dado un paradero.

Además, debes tener cuidado con la solicitud para obtener el tiempo de llegada de los próximos buses, debido a que puede que no sea exitoso. En ese caso, retornaría lo siguiente:


```
{"title": "smsbus webservice timeout"}
```


Si ocurre esto, deberás volver a realizar el *request* hasta que sea exitoso. Por ejemplo, un resultado exitoso realizado a la ruta `https://api.scltrans.it/v2/stops/PB8/next_arrivals`, debería verse así:


```
{"results":
    [
        {
            "bus_plate_number": "CJRF-76",
            "direction_id": 0,
            "calculated_at": "2018-11-16 23:04",
            "arrival_estimation": "En menos de 10 min.",
            "route_id": "107",
            "is_live": true,
            "bus_distance": "26"
        },
        {
            "bus_plate_number": "CJRF-80",
            "direction_id": 1,
            "calculated_at": "2018-11-16 23:04",
            "arrival_estimation": "Entre 15 Y 23 min. ",
            "route_id": "107",
            "is_live": true,
            "bus_distance": "5016"
        },
        {
            "bus_plate_number": "CJRF-49",
```

```
            "direction_id": 0,
            "calculated_at": "2018-11-16 23:04",
            "arrival_estimation": "En menos de 10 min.",
            "route_id": "101",
            "is_live": true,
            "bus_distance": "2044"
        },
        {
            "bus_plate_number": null,
            "direction_id": 1,
            "calculated_at": "2018-11-16 23:04",
            "arrival_estimation": "No hay buses que se dirijan al paradero.",
            "route_id": "B22",
            "is_live": true,
            "bus_distance": null
        },
        {
            "bus_plate_number": null,
            "direction_id": null,
            "calculated_at": "2018-11-16 23:04",
            "arrival_estimation": "Servicio fuera de horario de operacion para ese paradero",
            "route_id": "107C",
            "is_live": true,
            "bus_distance": null
        }
    ]
}
```


4. Expresiones regulares

Cada persona que quiera utilizar las funciones de DCConnect deberá utilizar un correo electrónico y una contraseña para acceder, por lo que deberás simular un inicio de sesión. Tu programa debe ser capaz de validar está información, utilizando sólo *expresiones regulares* (también conocidas en inglés como *regexes*) para validar los siguientes requerimientos. Para esto, recuerda que **debes** usar el módulo `re` que viene *built-in* en Python.

Para el correo electrónico,

- no contiene caracteres que no sean letras, números, guiones bajos (`_`) y arrobas (`@`);
- contiene exactamente un arroba;
- antes de la arroba, la cantidad de caracteres debe estar entre 3 y 8;
- y después de la arroba, esta cantidad debe estar entre 4 y 12 y contiene exactamente un punto.

Y para la contraseña,

- la cantidad de caracteres debe estar entre 8 y 12;
- debe contener al menos una mayúscula;
- y sólo contiene caracteres alfanuméricos.

Tanto para la dirección de correo como para la contraseña, sólo pueden aparecer caracteres que estén en ASCII. Además, si todas estas condiciones se cumplen, entonces el usuario ha escrito una dirección de correo y una contraseña válida, permitiendo el inicio de sesión.

5. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.6.
- Si no se encuentra especificado en el enunciado, asume que el uso de cualquier librería Python está prohibido. Pregunta en la *issue* especial del foro si es que es posible utilizar alguna librería en particular.
- Debes adjuntar un archivo `README.md` **conciso y claro**, donde describas los alcances de tu programa, cómo correrlo, las librerías usadas, los supuestos hechos, y las referencias a código externo. **Tendrás hasta 24 horas después del plazo de entrega** de la tarea para subir el *readme* a tu repositorio.
- Tu tarea podría sufrir los descuentos descritos en la **guía de descuentos**.
- Entregas con atraso de más de 24 horas tendrán calificación mínima (1,0).
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.

Las tareas que no cumplan con las restricciones del enunciado obtendrán la calificación mínima (1,0).