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

IIC2233 Programación Avanzada (2020-1)

Evaluación Recuperativa

Entrega

- **Fecha y hora:** lunes 13 de julio de 2020, 20:00
- **Lugar:** Repositorio personal de GitHub — Carpeta: `Recuperativa`

Importante: Nota que la carpeta de entrega es una que no existe actualmente en tu repositorio personal. Por eso debes crear la carperta `Recuperativa`, al mismo nivel que las carpetas `Tareas` y `Actividades`, y hacer y entregar tú desarrollo de la evaluación en ella.

Objetivos

- Tener la oportunidad de obtener bonificación en calificaciones del curso.
- Aplicar variados conocimientos del curso para la creación de una herramienta de utilidad.

Índice

1. Introducción

Las vacaciones se acercan y no puedes esperar por tomar tus ramos favoritos del DCC para el siguiente semestre. Desafortunadamente, descubres que el sitio usual de búsqueda de cursos fue *hackeado* por la malvada gatochico, disminuyendo considerablemente la oferta de vacantes disponibles para los cursos que más querías tomar. Para solucionar este inconveniente, deseas crear *DCCursos* y con eso planificar diversas opciones de horario, y así no atrasarte por culpa de gatochico.

2. Flujo del programa

DCCursos es un simulador de búsqueda de cursos y armado de horario semestral. Se interactúa con este programa mediante un formulario gráfico que permite buscar cursos mediante múltiples filtros. Al usar el programa, el usuario podrá obtener la información que desee de distintos cursos para posteriormente programar un horario.

En cuanto a implementación, *DCCursos* se separa en dos partes: **el cargado de información** de los distintos cursos, y la **implementación de consultas** sobre los datos de cursos. El resto del programa, como la interfaz gráfica y los programas de pruebas, **se entregan implementados** y podrás usarlos para poner a prueba tu implementación.

3. Cursos y base de datos

3.1. Representación de cursos

Los cursos son los ramos universitarios que un usuario puede buscar y eventualmente agregar a su horario. Tu programa representará internamente cada entidad de curso como un **diccionario** (`dict`), que sigue el siguiente formato general de llave-valor:

Llave	Tipo de valor	Descripción	Ejemplo de valor
`"Nombre"`	`str`	Nombre del curso.	`"Programación Avanzada"`
`"Retiro"`	`str`	Indica si el curso es retirable o no. Puede ser `"SI"` o `"NO"`.	`"SI"`
`"Aprobacion Especial"`	`str`	Indica si el curso tiene aprobación especial. Puede ser `"SI"` o `"NO"`.	`"NO"`
`"Creditos"`	`int`	Créditos del curso.	`10`
`"Ingles"`	`str`	Indica si el curso de dicta en inglés. Puede ser `"SI"` o `"NO"`.	`"NO"`
`"Sigla"`	`str`	Sigla del curso.	`"IIC2233"`
`"Prerrequisitos"`	`list`	Indica los prerrequisitos del curso. Es una lista de *strings* con las siglas de cada prerrequisito.	`["IIC1103", "IIC1102"]`
`"Secciones"`	`dict`	Diccionario con las secciones del curso. Cada llave es un *string* con el número de sección, y el valor asociado es otro diccionario, cuyo formato se explica más abajo.	`{"1" : {...}, ...}`

Cada sección del curso será otro **diccionario** (`dict`), que sigue el formato llave-valor:

Llave	Tipo de valor	Descripción	Ejemplo de valor
`"Campus"`	`str`	Campus donde se dicta.	`"San Joaquín"`
`"Formato"`	`str`	Formato en que se dicta.	`"Presencial"`
`"Modulos"`	`dict`	Diccionario con la distribución de módulos de un curso.	`{"AYU": [("M", 4)], ...}`
`"NRC"`	`str`	Código NRC de la sección.	`"10760"`
`"Profesor"`	`str`	Profesor(a) de la sección.	`"Florenzano Fernando"`
`"Seccion"`	`str`	Sección del curso.	`"1"`
`"Semestre"`	`str`	Semestre en que se dicta el curso.	`"2020-1"`
`"Vacantes disponibles"`	`int`	Vacantes disponibles de la sección.	`16`
`"Vacantes totales"`	`int`	Vacantes totales de la sección.	`100`

A su vez, cada diccionario de módulos es similar, tiene siete llaves, una por tipo de módulo, y el valor asociado es una lista con los horarios asociados al tipo de módulo correspondiente. Los horarios se representan como una tupla de largo dos, donde la primera posición es un carácter (`str`) que representa el día, y la segunda es un número (`int`) que representa el módulo horario. Si la lista asociada está vacía, es porque no hay horarios asociados a ese tipo de módulo.

Los posibles valores de llaves son `"AYU"` (ayudantía), `"CLAS"` (clase), `"LAB"` (laboratorio), `"PRA"` (práctica), `"TAL"` (taller), `"TER"` (terreno) o `"TES"` (tesis). Los valores de caracteres de día son `"L"`, `"M"`, `"W"`, `"J"` y `"V"`. Y los valores de módulo horario van entre `1` y `8`.

A modo de ejemplo, el diccionario de módulos para el curso Programación Avanzada es el siguiente:

```
{
    "AYU": [("M", 4)],
    "CLAS": [("J", 4), ("J", 5)],
    "LAB": [],
    "PRA": [],
    "TAL": [],
    "TER": [],
    "TES": []
}
```

3.2. Cargar los datos

La información de los distintos cursos que tu programa podrá mostrar se te provee como archivos en la carpeta `datos`, que debes colocar en la carpeta de trabajo `Recuperativa`. En ella hay tres archivos en formato JSON: `2019-1.json`, `2019-2.json` y `2020-1.json`.

Cada uno de estos archivos corresponde a los datos con todos los cursos del DCC de un año y semestre específico. Tu programa siempre funcionará en base a uno de esos archivos, y lo hará llamando la función `def cargar_cursos(semestre)` en `cargar_cursos.py` que **debes completar**.

Esta función, en base al `semestre` entregado como argumento, cargará el archivo JSON correspondiente y transformará los datos cargados a la representación de diccionarios especificada en Representación de cursos. Finalmente debe retornar un único gran diccionario que contiene todos los cursos cargados. Las llaves de este diccionario deben ser *strings* con las siglas de los cursos, y los valores son la representación de diccionario de cada curso, anteriormente descrita.

Para procesar el contenido de un archivo JSON y transformarlo al gran diccionario pedido, hay que tener varias consideraciones presentes. Primero, el contenido de cada archivo JSON tiene una estructura similar al diccionario objetivo. Como llaves hay siglas de cursos, que como valores tienen diccionarios que representan cursos individuales. Pero, hay algunos atributos que les falta procesamiento para que se coincidan con Representación de cursos. Específicamente:

- Los valores asociados a `"Prerrequisitos"` contienen un *string* con las siglas de cursos prerrequisitos separados por `";"` y contienen siglas de cursos que no son del DCC (siglas que no comienzan con IIC). Es necesario transformar el valor en una lista de prerrequisitos que solo contengan cursos IIC.
- Los valores asociados a `"Modulos"` son también *strings* que codifican la información de módulos de los cursos. Es necesario transformar el valor utilizando la función `traducir_modulos`, que se detalla más abajo.

A continuación se muestra como se ve el curso IIC2233 Programación Avanzada en uno de los archivos JSON base, considerando solo la sección 1:

```
{
    "IIC2233": {
        "Nombre": "Programación Avanzada",
        "Retiro": "SI",
        "Aprobacion Especial": "NO",
        "Creditos": 10,
        "Ingles": "NO",
        "Sigla": "IIC2233",
        "Prerrequisitos": "IIC1103;IIC1102",
        "Secciones": {
            "1": {
                "Campus": "San Joaquín",
                "Formato": "Presencial",
                "Modulos": "AYU#M:4;CLAS#J:4,5;LAB;PRA;TAL;TER;TES",
                "NRC": "10760",
                "Profesor": "Florenzano Fernando",
                "Seccion": "1",
                "Semestre": "2020-1",
                "Vacantes disponibles": 16,
                "Vacantes totales": 100
            }
        }
    }
}
```

Como mencionado, existe la función `def transformar_modulos(string_modulos)` **que debes completar**. Esta función recibe un *string* que codifica los módulos horarios de un curso, y retorna la representación de diccionario de dichos módulos especificada en Representación de cursos.

El formato de dichos *strings* es el siguiente: se separará el tipo de módulo por `";"`. Por tipo de módulo se separará su identificador de tipo por un `"#"`. Para describir un horario, se encuentra primero el día mediante su carácter identificador, y separado por `":"` se encuentra el número de módulo horario. En el caso de que más de un día comparta el mismo módulo, los días se separarán por `","`, y lo mismo ocurrirá si más de un módulo horario comparte el mismo día, por lo que se separarán los módulos por `","`.

Siempre se incluyen los siete tipos de horario en el *string*, pero no todos contendrán módulos de clases. Un ejemplo de esto se puede observar en el valor asociado a `"Modulos"` en la figura anterior para Programación Avanzada. Ese ejemplo muestra el caso en que un día tiene más de un módulo horario, pero también puede ocurrir que múltiples días compartan un módulo. Como por ejemplo:

```
"Modulos": "AYU;CLAS#L,W,V:3;LAB;PRA;TAL;TER;TES"
```

En este ejemplo los días lunes, miércoles y viernes comparten el módulo 3 de clases, y no tiene módulos en otro tipo. En cuyo caso debe convertirse a un diccionario como el siguiente:

```
{
    "AYU": [],
    "CLAS": [("L", 3), ("W", 3), ("V", 3)],
    "LAB": [],
    "PRA": [],
    "TAL": [],
    "TER": [],
    "TES": []
}
```

Como mencionado, todo lo relacionado a cargado de cursos se debe implementar en el módulo `cargar_cursos.py`. Puedes hacer prueba del funcionamiento de tus funciones ejecutando directamente ese módulo. Una vez implementado el cargado de datos, podrás avanzar a la sección de implementación de consultas, que se explica a continuación.

4. *DCCursos*

DCCursos es la entidad del programa que simula el armado correcto de cursos para un usuario, y se utiliza como la clase `DCCursos`, definida en `DCCursos.py`. Posee como atributo un diccionario `horario`, que comienza vacío y se va rellenando a medida que el usuario agregue ramos. También tiene un diccionario `cursos`, el cual contendrá todos los cursos de un semestre.

Esta clase posee diversos métodos que **ya vienen implementados**, los cuales se encargaran de llamar y procesar el resultado de las consultas que se explican a continuación. Los atributos y funciones de esta clase **no deben ser modificados**.

4.1. Consultas

La entidad *DCCursos* hace uso de una serie de filtros y consultas sobre los datos de cursos para funcionar. Estas funcionalidades se listan como funciones en `consultas.py` y **debes completarlas**. Todas las funciones reciben al menos un argumento diccionario `dicc_de_cursos` que contiene cursos cargados en el formato explicado en Representación de cursos.

Cada función debe funcionar suponiendo que todos los cursos disponibles se proveen a través de `dicc_de_cursos`, y debe retornar otro diccionario que sigue la misma estructura, pero con información filtrada. Por ejemplo, si un curso no cumple la condición del filtro específico, entonces no existe su sigla ni su información en el diccionario que retorna. En algunas situaciones, pueden existir cursos donde solo algunas secciones cumplen los requisitos. En ese caso, el curso si debe aparecer en el resultado de la consulta, pero solo debe contener aquellas secciones que cumplan los requisitos.

Deberás implementar las siguientes consultas:

- `def filtrar_por_prerrequisitos(curso, dicc_de_cursos)`: Esta consulta recibe un diccionario de curso en `curso`, y retorna un diccionario con todos los cursos en `dicc_de_cursos` que son prerrequisitos de `curso`.
- `def filtrar_por_cupos(cupos, dicc_de_cursos)`: Este filtro retorna un diccionario con todos los cursos con secciones que tengan más o la misma cantidad de vacantes disponibles dadas por el `int` recibido en `cupos`. Secciones que no tengan vacantes libres no se deben retornar en el diccionario.
- `def filtrar_por(llave, string, dicc_de_cursos)`: Recibe una `llave` (`str`) que representa un atributo de curso y un *string*. La llave puede ser únicamente `'Nombre'`, `'Profesor'`, `'NRC'` o `'Sigla'`. Este filtro retorna un diccionario con todos los cursos de `dicc_de_cursos` que contengan a `string` en el campo asociado a `llave`, de forma insensible a mayúsculas y minúsculas. Si se filtra por NRC o Profesor, los cursos solo deben incluir las secciones que coinciden con el criterio de búsqueda. En caso de que `llave` no sea ninguno de los valores posibles, entonces el filtro debe retornar un diccionario vacío.
- `def filtrar_por_modulos(modulos_deseados, dicc_de_cursos)`: Esta consulta recibe módulos horarios en la lista `modulos_deseados`, donde cada elemento es una tupla de la forma `(dia, mod)`, donde `dia` es un `str` asociado al día y `mod` es un `int` asociado al módulo (ej. `('L',1)` es el día lunes módulo 1, como detallado en Representación de cursos). Este filtro retorna un diccionario con todos los cursos que tengan secciones con alguna actividad durante esos módulos. Es decir, secciones de cursos que no tengan actividades esos módulos no se deben incluir.
- `def filtrar_por_cursos_compatibles(horario, dicc_de_cursos)`: Esta consulta recibe una lista `horario` que contiene los códigos NRC de secciones (`str`) y retorna un diccionario con todos los cursos en `dicc_de_cursos` y secciones que **no tengan topes de horario** con los cursos especificados en `horario`. Para obtener la información horaria de los cursos especificados en `horario`, utiliza `dicc_de_cursos`.

Para probar tus implementaciones con ejemplos simples puedes ejecutar el módulo `consultas.py`. Siéntete libre de alterar los casos entregados, y también de definir funciones auxiliares si así lo estimas conveniente.

5. Poner *DCCursos* a prueba

5.1. Programa con interfaz gráfica

Como fue mencionado anteriormente, te entregamos una interfaz gráfica para que puedas probar de forma más intuitiva y visual tu programa y sus distintas funciones. Para funcionar de manera correcta, la interfaz está compuesta por tres partes:

- **Búsqueda de cursos:** Acá el usuario puede seleccionar un semestre e ingresar los distintos filtros que se le aplicaran a los cursos. Después de ingresar uno o más de ellos, puede buscar los cursos que coincidan con los filtros o limpiar el formulario y los resultados obtenidos.
- **Resultados de la Búsqueda:** Acá se muestran los distintos cursos que cumplen con los filtros indicados anteriormente. Cada curso cuenta con un botón que le permita agregar el curso al horario.
- **Mi Horario:** Esta sección cuenta de tres partes:
  - Un horario, el cual señala la disposición de las clases.
  - Un listado de cursos, donde se nombra cada curso tomado por el usuario. Además, cuenta con la opción de eliminar cursos del horario, para esto basta con hacer *click* sobre el curso que se desea eliminar.

- El botón "Buscar Cursos Compatibles", el cual permite buscar cursos compatibles con el horario que actualmente tiene el usuario.

Para hacer uso de la interfaz gráfica, basta con ejecutar el archivo `main.py` al interrior de tu carpeta `Recuperativa`, y también contar con la carpeta `GUI` y archivo `DCCursos.py` al mismo nivel. **No debes modificar ninguno de archivos en `GUI` ni `main.py`**.

5.2. *Tests* automáticos

Junto con el código de esta tarea, se incluye el módulo `test.py`, que contiene a la clase `Test`. Esta clase ya viene implementada, y puedes modificarla para hacer tus propias pruebas sobre el desarrollo de tus funciones. Puedes cambiar sus métodos directamente, agregar atributos, heredar de ella, etc. Si bien esta clase no es parte de lo que debes completar, esperamos que te ayude durante tu desarrollo para probar tu código y para encontrar posibles errores. Además, viene con un ejemplo de cómo ejecutaremos *tests* en el mismo archivo.

La clase `Test` recibe una lista de argumentos, una lista de *outputs*(resultados) esperados y la función a probar. Una vez instanciada la clase, el método `probar_casos` recorre la lista de argumentos, utiliza la función a evaluar para obtener el *output* de esta, y lo compara con el *output* esperado. Si ocurre una excepción durante alguno de estos pasos, el resultado se considera incompleto o incorrecto. **Solo si el resultado obtenido coincide con el esperado, el *test* se considerará correcto**.

6. Corrección (automática) de evaluación

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 de un ayudante. Para lograr esto, utilizaremos archivos de *test* que ejecutarán tu código, y a partir de este, obtener un *output* definido. Si este *output* coincide con el de pauta, se considera ese test como correcto.

Para cada función, se realizarán múltiples y variados *tests*, de distinta complejidad, que buscarán detectar su correcta 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.

A continuación se listan las funciones que serán corregidas, y los puntajes asociados a cumplir el 100 % de sus *tests* automáticos:

- Funciones en `cargar_cursos.py`.
  - (1.0 pt.) `traducir_modulos`.
  - (2.0 pts.) `cargar_cursos`.
- Funciones en `consultas.py`.
  - (1.0 pt.) `filtrar_por_prerrequisitos`.
  - (1.0 pt.) `filtrar_por_cupos`.
  - (1.5 pts.) `filtrar_por`.
  - (1.5 pts.) `filtrar_por_modulos`.

- (2.0 pts.) `filtrar_por_cursos_compatibles`.

En total, se pueden acumular 10 puntos por una implementación completamente correcta. Esto se traduce en una bonificación décimas sobre el promedio de una de las áreas del curso: Actividades o Tareas. Específicamente, se aplicará sobre el promedio que sin bonificación sea **más bajo** y la cantidad de décimas será proporcional a los puntos obtenidos bajo la siguiente fórmula:

$$n = \begin{cases} \mathbf{P} \times 0{,}5 & \text{si } \mathbf{T} \leq \mathbf{AC} \\ \mathbf{P} & \text{si } \mathbf{T} > \mathbf{AC} \end{cases}$$

Donde n es la cantidad de décimas, **P** es la cantidad de puntos obtenidos en la corrección automática, **T** es el promedio ponderado acumulado de Tareas y **AC** es el promedio de Actividades, calculado como descrito en el programa y ***syllabus***.

A modo de ejemplo, si los promedios de un estudiante en Tareas y Actividades son `3.90` y `4.75`, y obtiene `8.25` puntos en la corrección automática de esta evaluación, entonces sus promedios terminarían en `4.31` (`3.90 + 0.4125`) y `4.75`, respectivamente. Notar que tal estudiante pasaría a aprobar el curso tras este bonificación, ya que previamente no cumple con el requisito de promedio mínimo en Tareas.

7. Consideraciones importantes de entrega

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 o filtro, es un buen momento para subir avance. Así, no se acumula mucho desarrollo nuevo para *pushs* cercanos a la hora de entrega.

Para esta evaluación **no se evaluará el uso de `.gitignore`**. De todas formas, lo único necesario que se necesita se entregue son los **módulos a completar: `consultas.py` y `cargar_cursos.py`**. Pero en caso de subir el resto de los módulos de apoyo, no hay problema.

Tampoco habrea 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 `Recuperativa`. Los módulos base se espera usen siguiendo la misma estructura que como entregado, al primer nivel de la carpeta `Recuperativa`:

```
Recuperativa
├── cargar_cursos.py
├── consultas.py
├── DCCursos.py
├── tests.py
├── main.py
├── datos
│   ├── 2019-1.json
│   ├── 2019-2.json
│   └── 2020-1.json
└── GUI
    └── ...
```

8. Restricciones y alcances

- Esta evaluación es **estrictamente individual**, y está regida por el **Código de honor de Ingeniería**.
- Tu programa debe ser desarrollado en Python 3.7.
- Tu programa debe estar compuesto por uno o más archivos de extensión `.py`.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería Python está prohibido. Pregunta en una *issue* del **foro** si es que es posible utilizar alguna librería en particular.
- Uso de PEP8 no se evaluará ni descontará.
- Entregas con atraso no se considerarán.
- 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).