



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 Programación Avanzada (I/2018)

Tarea 6

Entrega

- Tarea obligatoria
 - **Fecha/hora:** domingo 24 de junio del 2018, 23:59 horas.
 - **Lugar:** GitHub – Carpeta: **Tareas/T06/**
- README.md obligatorio
 - **Fecha/hora:** lunes 25 de junio del 2018, 23:59 horas.
 - **Lugar:** GitHub – Carpeta: **Tareas/T06/**

Objetivos

- Aplicar conocimientos de creación de redes para comunicación efectiva entre computadores.
- Diseñar e implementar una arquitectura cliente-servidor.
- Aplicar conocimientos de manejo de *bytes* para crear o modificar un tipo de archivo preestablecido.

Índice

1. Introducción	3
2. <i>PrograBand</i>	3
3. Manejo de <i>bytes</i> (formato MIDI)	3
3.1. <i>Header</i>	4
3.2. Canal	5
3.3. Notas y silencio	7
3.4. Evaluar MIDI (importante)	7
4. Redes (<i>networking</i>)	8
4.1. Servidor	8
4.2. Cliente	9
4.3. Estructura de archivos	9
5. Interfaz	10
5.1. Ventana de inicio	10
5.1.1. Usuario	10
5.1.2. Canciones en edición (o ventanas de edición abiertas)	10
5.1.3. Canciones listas	10
5.1.4. Acciones para una canción lista	10
5.1.5. Nueva canción	11
5.2. Ventana de edición y/o espectador	11
5.2.1. Sección de edición	12
5.2.2. Sección de notas	13
5.2.3. Sección conectados	13
5.3. Evaluar Interfaz (Importante)	13
6. <i>Bonus</i>	14
6.1. Notas simultáneas (5 décimas)	14
6.2. <i>Chat</i> (5 décimas)	14
7. Anexo: PEP8	14
8. Restricciones y alcances	14

1. Introducción

Después del exitoso resultado que tuvo *Code with Fire* en el mercado, decides explorar en otras áreas en busca de un desafío. En eso, te encuentras con un proyecto interesante llamado *PrograBand*. Esta es una asociación de artistas compositores que ha estado trabajando en la evolución de su sistema de creación de melodías. Durante el paso de los años, ellos han buscado revolucionar la manera de compartir y crear música, sin tener éxito en esta misión.

Es por esto que como estudiante de *Programación avanzada*, te las ingenias para integrarte en el equipo y buscar una manera de generar melodías exitosas, de calidad y fácilmente compartibles por medio de una interfaz gráfica amigable.

Sin embargo, luego de llevarle esta idea al jefe de la asociación, éste no se convenció ya que era demasiado simple respecto a sus expectativas. Debido a esto, se te ocurrió que, además de todo lo que tenías pensado, tendrás la opción de editar y compartir canciones con tus amigos a través de internet.

¡Es tu momento de demostrar lo que sabes!

2. *PrograBand*

Esta tarea consiste en desarrollar un programa de creación y edición de música en tiempo real en formato MIDI. Cada usuario tiene la capacidad de entrar a una ventana de inicio, donde se puede elegir el nombre de usuario, además de ver la lista de canciones terminadas y/o en edición, o crear una nueva canción. En la ventana de edición, se pueden agregar o eliminar notas a la canción seleccionada para cambiar su melodía, en paralelo con varias personas a la vez que se encuentran viendo la misma canción.

3. Manejo de *bytes* (formato MIDI)

MIDI (sigla de *Musical Instrument Digital Interface*) es un protocolo de comunicación que permite a las computadoras, sintetizadores y otros dispositivos electrónicos musicales intercambiar información.

Un archivo MIDI contiene una serie de instrucciones que el sintetizador u otro generador de sonido utiliza para reproducir un sonido en tiempo real. Estas instrucciones son mensajes que indican al instrumento cuáles son las notas musicales, su duración, la fuerza de toque y las modulaciones de los parámetros de los sonidos; es decir, toda la información necesaria para la generación de sonidos. Cada sonido emitido por un MIDI está formado por paquetes de instrucciones en formato numérico, en otras palabras, en formato de *bytes*.

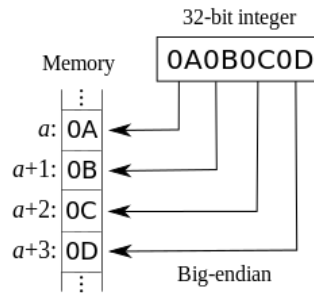
En un archivo MIDI se pueden identificar **dos** tipos de *chunks*: el *header* y un canal. El *header* es una secuencia de *bytes* que describe la *metadata* de un archivo, mientras que un canal representa una secuencia de notas a generar. Un canal solo permite generar un sonido a la vez, por lo tanto, si se quiere simular un sonido de dos o más notas al mismo tiempo, es necesario utilizar más de un canal. En este caso, un archivo MIDI consiste en un *header* con muchos canales, sin embargo, para esta tarea se utilizarán archivos MIDI de un solo canal.

El formato estándar de un *chunk* es:

Type	Length	Data
------	--------	------

- *Type* — Esta sección ocupa 4 *bytes* y almacena el nombre del *chunk*. Para el *header*, el *type* es `b'MThd'` y para un canal, el *type* es `b'MTrk'`.

- *Length* — Esta sección ocupa 4 *bytes* y almacena el largo en *bytes* de la sección *Data*. Estos *bytes* están en formato *big endian*¹. El orden de *endianness* indica cómo se ordenan los *bytes* de un número cuando éste se representa usando más de un *byte*. Con *big endianness*, en esta lista de 4 *bytes*, el primer *byte* contiene los *bits* más significativos del número. A continuación, se presenta una imagen que muestra cómo un número de 32 *bits* fue dividido en 4 elementos de 8 *bits* (análogo a dividir un número de 4 *bytes* en 4 elementos de 1 *byte*).



Para transformar la representación de un número a *bytes*, se recomienda utilizar el método de Python `to_bytes` donde uno de sus argumentos debe ser `byteorder="big"`.

- *Data* — Esta sección ocupa tantos *bytes* como los indicados en *Length*. Esta secuencia almacena instrucciones, cuyo formato depende del tipo de *chunk*.

3.1. Header

El *header* indica la cantidad de canales que tendrá la melodía y los *ticks* necesarios para tocar una corchea (cuarto de tiempo musical). En base a estos *ticks* es posible definir las siguientes notas como la corchea, blanca, etcétera.

La parte de *Data* del *header* se compone de tres números, cada uno de ellos representado con 2 *bytes* en orden *big endian*.

1. Formato del MIDI. Contiene uno de los siguientes valores:
 - 0. Archivo con un único canal.
 - 1. Archivo con múltiples canales.
 - 2. Archivo con múltiples canciones. Por ejemplo, múltiples archivos de formato 0.

En esta tarea solo se usará el formato 1.

2. Número de canales. Indica la cantidad de canales que vendrán después. Esto se puede considerar como la máxima cantidad de notas que estarán sonando de forma simultánea. En esta tarea, será 1 si no se realiza el *bonus*.
3. Número de *ticks* por un tiempo musical. Este número representa el tiempo necesario que el computador le otorga a “un tiempo musical” en la melodía. Por defecto, esta tarea debe usar 160 *ticks* por tiempo musical.² Esto se explica con más detalle en la sección 3.2.

¹Para saber más: <https://es.wikipedia.org/wiki/Endianness>

²No es necesario saber cuánto tiempo, en segundos, es un *tick*.

3.2. Canal

Cada canal permite tocar una secuencia de notas o silencios. Para esto existen dos mensajes distintos: *Note On* y *Note Off*. Como sus nombres lo indican, estos mensajes representan, respectivamente, la acción de empezar a emitir y la acción de detener un sonido. Ambos mensajes tienen la misma estructura que se compone de los siguientes elementos:

Tiempo	Tipo de mensaje	Nota	Intensidad
--------	-----------------	------	------------

- **Tiempo** — Este número representa el número de *ticks* para ejecutar la acción. Cuando el mensaje es tipo *Note On*, el tiempo es 0. Cuando es *Note Off*, este número será el número de *ticks* necesarios según la duración de la nota. Como se mencionó anteriormente, en esta tarea se usará 40 *ticks* para representar **1/4 de Tiempo**. Una negra, que dura **1 Tiempo**, equivaldrá a 160 *ticks*. Con dicha información, la duración de cada nota es:


NOMBRE	FIGURA	SILENCIO	VALOR
Redonda			4 Tiempos
Blanca			2 Tiempos
Negra			1 Tiempo
Corchea			1/2 Tiempo
Semicorchea			1/4 Tiempo
Fusa			1/8 Tiempo
Semifusa			1/16 Tiempo

Figura 1: Duración de las notas. Si 1 de tiempo son 160 *ticks*, una corchea (1/4 tiempo) usará 40 *ticks*.

Las notas pueden, además, poseer un puntillo³, el cual hace que su duración sea igual a 1,5 de su duración original. Por ejemplo, una negra dura 1 tiempo (160 *ticks*), y una negra con puntillo dura 1 tiempo y medio (160 × 1,5). En caso de una corchea, que dura medio tiempo, con puntillo durará 3/4 de tiempo (160 × 3/4).

Tras definir la duración de la nota (*e.g.* negra, corchea, corchea con puntillo, blanca), es necesario codificar su tiempo en una secuencia de *bytes*. Para esto hay que seguir los siguientes pasos:

1. Transformar el número a binario (sólo 0 y 1). Para esto pueden ocupar `format(número, 'b')`. Por ejemplo, si el número es 641, en binario sería 1010000001 (usando `format(641, 'b')`).
2. **De derecha a izquierda**, debes generar fragmentos de a 7 *bits* y pasarlos a una lista usando *big endianness*, es decir, el primer fragmento que generas usará el último lugar de la lista y el último

³Un puntillo se puede señalar colocando un punto (".") al lado de la nota, por ejemplo, .

fragmento que generas estará en el primer lugar de la lista. Es posible que el último fragmento generado no ocupe justamente 7 bits, en ese caso se debe rellenar (*padding*) con ceros. Por ejemplo, para el número 641, que en binario es 1010000001, se separa de a 7 bits quedando de la siguiente forma [101, 0000001]. El primer elemento (o último fragmento obtenido) usa menos de 7 bits así que se rellena con ceros. Por lo tanto, la lista final es: [0000101, 0000001]

3. **De derecha a izquierda**, a cada elemento de la lista, menos el primero, se le agrega un octavo bit al inicio con valor 1, y al primer elemento, se le agrega un octavo bit al inicio con valor 0. Con el ejemplo anterior, la lista era: [0000101, 0000001]. Con estos nuevos bits, ahora queda: [10000101, 00000001]
 4. Cada elemento de la lista ahora representa un *byte*. Dependiendo del número a transformar, esta lista puede contener 1, 2 o más *bytes*. Como podrás ver, el largo de esta lista va depender del número que se quiere transformar. Por lo tanto, no puedes suponer que cada segmento dentro de *Data* tiene la misma cantidad de *bytes*.
- **Tipo de mensaje** — Este número indica si el mensaje es del tipo *Note On* o *Note Off*. En caso de ser *Note On*, el número debe ser 144. En el otro caso, de ser *Note Off*, el número debe ser 128. Este número se representa en 1 *byte*.
 - **Nota** — Este número indica qué nota se está generando. Por ejemplo, un *DO*, *RE*, *Sib*. Dicho número se representa en 1 *byte* y, según la nota y octava, tendrá un valor de acuerdo a la figura 2. Esta figura está con la nomenclatura en ingles, pero en la tabla 1 de la sección 5.2.1 se indica la equivalencia entre la nomenclatura en inglés y en español.

Octava	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	0	1	2	3	4	5	6	7	8	9	10	11
1	12	13	14	15	16	17	18	19	20	21	22	23
2	24	25	26	27	28	29	30	31	32	33	34	35
3	36	37	38	39	40	41	42	43	44	45	46	47
4	48	49	50	51	52	53	54	55	56	57	58	59
5	60	61	62	63	64	65	66	67	68	69	70	71
6	72	73	74	75	76	77	78	79	80	81	82	83
7	84	85	86	87	88	89	90	91	92	93	94	95
8	96	97	98	99	100	101	102	103	104	105	106	107
9	108	109	110	111	112	113	114	115	116	117	118	119
10	120	121	122	123	124	125	126	127				

Figura 2: Nota en nomenclatura inglesa y octava a valor numérico

- **Intensidad** — Este número indica la fuerza utilizada para generar el sonido. Esto repercute en el volumen que tendrá la nota. En la figura 3 se muestra el valor de todas las intensidades junto al valor numérico. Este número se representa en 1 *byte*.

<i>pppp</i>	= 8
<i>ppp</i>	= 20
<i>pp</i>	= 31
<i>p</i>	= 42
<i>mp</i>	= 53
<i>mf</i>	= 64
<i>f</i>	= 80
<i>ff</i>	= 96
<i>fff</i>	= 112
<i>ffff</i>	= 127

Figura 3: Intensidades posibles y valor numérico.

Finalmente, luego de ingresar todas las notas y silencios, a este canal se le agregan cuatro valores de cierre. Estos son [0, 255, 47, 0]. Cada número debe ser pasado a un *byte*.

3.3. Notas y silencio

Para generar un sonido, es necesario mandar dos tipos de mensajes. Primero se manda un mensaje tipo *Note On* con tiempo igual a 0, el valor numérico de la nota y la intensidad deseada. Luego se manda un mensaje tipo *Note Off* con tiempo igual a la duración de la nota, el valor numérico de la nota y la misma intensidad usada en el primer mensaje.

Para generar un silencio, sólo es necesario mandar un mensaje de tipo *Note Off* con tiempo igual a la duración del silencio. El valor numérico de la nota y la intensidad deben ser 0.

3.4. Evaluar MIDI (importante)

En esta tarea se va a evaluar el **manejo de *bytes*** en cuatro diferentes ítems:

- Escritura de MIDI con notas y sin silencios.
- Escritura de MIDI con notas y silencios.
- Lectura de MIDI con notas y sin silencios.
- Lectura de MIDI con notas y silencios.

Para tener el puntaje en cada ítem, su código debe realizar todo el proceso de forma exitosa. **No se dará puntaje intermedio** por realizar algunas partes del ítem, pero otras no. Por ejemplo: leer *header*, pero no el canal; aceptar algunas notas y otras no, etcétera.

En caso de no realizar completamente la interfaz para poder crear un MIDI, se recomienda crear:

- `generar_midi.py` — Cuando se ejecuta, crea una canción en formato MIDI que puede ser reproducida en el computador. De esta forma, el ayudante podrá editar dichas notas para crear las canciones que haría en la interfaz.

- `leer_midi.py` — Cuando se ejecuta, lee un archivo MIDI e imprime en pantalla la información del *Header* y de cada nota y silencio. De esta manera, el ayudante podrá cargar los archivos MIDI de prueba y evaluar la lectura efectiva.

No puede suponer que cada *ítem* vale lo mismo y no puede utilizar alguna librería para manejar los MIDI. Todo debe ser hecho con manejo de *bytes*.

4. Redes (*networking*)

La implementación de *PrograBand* **debe** basarse en una arquitectura *cliente-servidor*, en la que todas las interacciones que se quieran realizar entre usuarios deberán contar con un servidor como mediador de la comunicación. Esta implementación tiene que usar un protocolo eficiente basado en el *stack* de protocolos **TCP/IP**.

Los requisitos de esta arquitectura son:

- Sólo se puede enviar y recibir mensajes de a lo más 1.024 *bytes*.
- Para cada mensaje a enviar, primero es necesario mandar otro mensaje previo que indique el tamaño del mensaje que se enviará.⁴ Luego, se debe mandar el paquete original fragmentado en segmentos de 1.024 *bytes*. Por ejemplo, si se quisiera mandar un MIDI de 3.596 *bytes*, se deberán mandar 5 mensajes. El primero donde se diga que el paquete es de 3.596 *bytes*, luego 3 paquetes de 1.024 *bytes* y finalmente un paquete de 524 *bytes* para formar el MIDI.

Su sistema deberá ser capaz de funcionar entre computadores que estén conectados a una misma red local. Para esto, necesitarán utilizar la librería **socket** y sus métodos. Las siguientes secciones describen cómo se deben implementar esta estructura.

4.1. Servidor

El servidor se debe encargar de la lógica del sistema, sus interacciones, el almacenamiento de datos de todos los archivos *subidos*, junto con la información de los usuarios y cualquier otra funcionalidad que consideren pertinente. Por lo mismo, no se debe implementar una interfaz gráfica para este componente, ya que funcionará como parte del *back-end*. Debe existir sólo una instanciación de este módulo en un único computador para poder ejecutar correctamente el programa. Además, en el módulo en el que escribirá el código correspondiente al servidor, **debe** colocar en las primeras dos líneas las variables **HOST** y **PORT** con sus valores respectivos, para que así al momento de corregir, sea fácil cambiar sus valores. Ambas variables **deben** ser utilizadas en el método **bind** del *socket* del servidor para que se realice la conexión.

Para poder conectarse por red local, deben:

- Estar conectados a una red,
- Y ajustar la variable **HOST** de su servidor a la IP local de su computador. Para saber cómo se obtiene la IP de tu computador, basta con buscar en Google⁵. No es difícil.

Como se indicó, el servidor no requiere de una interfaz gráfica, pero sí debe hacer **print** cada vez que:

- Se conecte un cliente. Debe darle algún nombre al cliente para identificarlo posteriormente en los siguientes **print**.
- Se desconecte un cliente. Debe indicar qué cliente es.

⁴Y no, no hay que enviar un mensaje previo al mensaje previo que indique cuántos *bytes* tiene el paquete previo.

⁵O bien, en DuckDuckGo, el buscador amigable.

- Servidor recibe un paquete de algún cliente. Debe indicar qué cliente envió el paquete, el tipo de paquete (*e.g.* indica su nombre de usuario, solicita alguna canción, envía alguna nota) y el tamaño del mensaje.
- Servidor manda un mensaje a algún cliente. Debe indicar qué cliente es, el tipo de mensaje (*e.g.* envía la canción solicitada, indica que nombre de usuario está bien, envía una nueva nota de la canción) y el tamaño del mensaje.

El formato de hacer los *print* queda a su criterio mientras exponga toda la información de forma ordenada, pero se recomienda el siguiente formato:

Cliente	Acción	Detalles
1	Conectarse	-
1	Enviar Mensaje	Enviando midi, 2.035 bytes
2	Conectarse	-
1	Desonectarse	-

4.2. Cliente

Este es el programa con el que interactuarán los usuarios, por lo que debes incluir la interfaz gráfica de *PrograBand*. El cliente tiene que ser capaz de conectarse al servidor para poder funcionar, el cual puede estar ejecutándose en el mismo computador, o en uno remoto. Si el cliente fuera ejecutado en dos computadores distintos en la misma red, ambos deberán poder interactuar de la misma forma con el servidor.

Al igual que en el caso del servidor, deberás escribir en las dos primeras líneas **HOST** y **PORT** junto con sus respectivos valores. Ambas variables **deben** ser usadas en el método **connect** del *socket* del cliente.

4.3. Estructura de archivos

La tarea se debe entregar con la siguiente estructura de carpetas y archivos:

```

T06
├── client
│   ├── main.py
│   └── (otros modulos del cliente)
├── server
│   ├── main.py
│   └── (otros modulos del servidor)
└── midis
    ├── midi1.mid
    ├── midi2.mid
    ├── midi3.mid
    ├── midi4.mid
    └── ...

```

Los programas del cliente y del servidor deben estar separados, cada uno en su propia carpeta. Además, como se indica, los archivos MIDI se deben encontrar dentro de la carpeta **midis**, que a su vez, está dentro de la carpeta **server**, donde se podrán agregar manualmente archivos y el servidor debe ser capaz de interpretarlos como corresponde permitiendo al cliente bajarlos y editarlos.

5. Interfaz

Para esta tarea deberás implementar una interfaz que será utilizada en el lado del cliente para la creación y edición de canciones. La interfaz también debe permitir bajar las canciones en formato MIDI.

5.1. Ventana de inicio

La ventana de inicio es la primera que ve el usuario al momento de abrir el programa. A continuación se muestra lo mínimo que debe contener dicha ventana:

El diagrama muestra la interfaz de usuario 'PrograBand' con los siguientes elementos:

- Un campo de entrada de texto etiquetado como 'Usuario' con el placeholder 'Ingrese su nombre...'.
- Una sección con dos paneles: 'Canciones en edición' y 'Canciones listas'.
- El panel 'Canciones en edición' contiene la lista: 'Waka Waka', 'Mi Corazon Encantado'.
- El panel 'Canciones listas' contiene la lista: 'Game of Thrones', 'Shingeki no Kyo'.
- Debajo de los paneles, una sección titulada 'Acciones para una canción lista' que contiene dos botones: 'Editar' y 'Descargar'.
- En la parte inferior, una sección titulada 'Nueva Canción' que incluye un campo de entrada de texto con el placeholder 'Ingrese nombre de la canción...' y un botón 'Crear Canción'.

5.1.1. Usuario

El usuario debe ser capaz de ingresar algún nombre de usuario. Este nombre debe cumplir con no poseer espacios y tener un mínimo de 6 caracteres. Esta verificación se debe hacer cuando el usuario solicita editar una canción, crear una canción y cuando se hace clic en una canción de la lista “Canciones en edición”.

5.1.2. Canciones en edición (o ventanas de edición abiertas)

Esta lista contiene todas las canciones que actualmente se están creando o editando. Cuando se hace clic en alguna de las canciones, el cliente debe verificar las condiciones del nombre de usuario, y el servidor debe verificar que el nombre de usuario sea único. En caso de cumplir ambas cosas, se debe desplegar la ventana de edición de la canción en modo espectador. En otro caso, se debe informar en la interfaz el motivo que no hay conexión. La forma de exponer dicha información queda a su criterio **mientras se use la interfaz y no la consola**.

5.1.3. Canciones listas

En esta lista están todas las canciones que se han creado en el servidor. Al momento de hacer clic sobre una de ellas, esta queda marcada para poder realizar alguna acción sobre ella.

5.1.4. Acciones para una canción lista

Luego de hacer *clic* sobre alguna canción de la lista de “Canciones listas”, el usuario puede:

- **Editar** — Con esta opción, el cliente verifica las condiciones del nombre de usuario, y el servidor verifica que sea único. En caso de cumplirla, la canción pasa de la lista “Canciones listas” a “Canciones en edición” y se abre la ventana de edición de canción en modo editor.
- **Descargar** — Con esta opción, el servidor le envía el MIDI al cliente y este se guarda con el nombre de la canción y extensión .mid. La carpeta donde se guarda queda a criterio de ustedes mientras lo especifiquen en el README.md. Para probar que todo salió bien pueden escuchar los MIDI con este enlace.

5.1.5. Nueva canción

El usuario debe ser capaz de crear nuevas canciones, para eso debe ingresar antes el nombre de la canción. Al momento de hacer clic en “Crear Canción”, el cliente debe verificar las condiciones del nombre de usuario y que el nombre de la canción tenga un mínimo de 6, y el servidor debe verificar que el nombre de la canción sea único. En caso de cumplir todo, esa canción se crea en el servidor.

Luego de crearse la canción en el servidor, se debe abrir una ventana correspondiente a la sala de edición para esa canción. Para aquellos usuarios que estén en la ventana de inicio al momento de la creación de la sala, les debe aparecer el nombre de esta en la lista de ventanas de edición abiertas.

5.2. Ventana de edición y/o espectador

En esta ventana es posible crear y editar una canción. Esta tiene dos modos: espectador y editor. En la primera opción, el usuario no podrá agregar nuevas notas, sino solo ver las personas conectadas y la “Lista de notas”, que se actualiza en tiempo real si algún otro usuario la edita. En la segunda opción, el usuario tiene acceso a agregar y eliminar notas. En la figura 4 se muestra lo mínimo que debe contener esta ventana.

Figura 4: Sala de edición

5.2.1. Sección de edición

En esta sección, si el usuario está en modo editor, él puede agregar una nota completando los siguientes parámetros:

- **Nota.** Se indica la nota musical a agregar, la cual puede ser en cualquiera del siguiente formato:

Nomenclatura español	Nomenclatura inglesa	Valor numérico
do	C	1
do#	C#	2
re	D	3
mib	Eb	4
mi	E	5
fa	F	6
fa#	F#	7
sol	G	8
sol#	G#	9
la	A	10
sib	Bb	11
si	B	12

Cuadro 1: Notas musicales. Importante: *Sib* == *la#*

- **Octava** — Es un número entre 0 y 10, ambos incluidos.
- **Intensidad** — Esta opción debe permitir al usuario ingresar la intensidad de la nota. El *input* puede ser en forma de *string* (pppp, ppp, pp, p, mp, mf, f, ff, fff o ffff) o en forma numérica cuyo número debe ser el indicado en la figura 3.
- **Duración** — Esta opción debe permitir al usuario ingresar la duración de la nota en forma de *string* o en forma numérica. Además, el usuario puede agregar un punto final si desea que la nota posea un puntillo. El *input* en forma de *string* o numérico se rigen bajo la siguiente tabla:

Duración	Valor numérico
Redonda	1
Blanca	2
Negra	3
Corchea	4
Semicorchea	5
Fusa	6
Semifusa	7

Cuadro 2: Duración notas

- **Silencio** — El usuario debe ser capaz de indicar si se va a ingresar una nueva nota o un silencio. El formato de como indicar tal diferencia queda a su criterio, pero **deben indicar en el Readme que criterio ocuparon, aunque sea intuitivo.**

Importante: todos los *input* que acepten *string* deben ser *case-insensitive*.⁶ Al momento de agregar la nota, el cliente debe verificar que:

- No hay espacios en los *inputs*.
- La nota es la indicada en la tabla 1.
- La nota y octava generan un valor numérico entre 0 y 127 según la figura 2.
- La duración es la indicada en el cuadro 2.
- La intensidad es alguna de las indicadas en la figura 3.

En caso de no respetar alguna regla, la interfaz debe avisar del error. La forma de exponer dicha información queda a su criterio **mientras se use la interfaz y no la consola.** En otro caso, el cliente envía el paquete con la información de la nota al servidor y ésta queda editada de forma inmediata en la canción.

El usuario que está editando también debe tener la opción de eliminar notas de su canción, esto debe pasar al hacer *clic* en cualquier nota de la lista.

5.2.2. Sección de notas

Cada vez que una nota es agregada a la canción, se debe añadir también a la lista de visualización. Además, el servidor debe procesar la adición para que todos los usuarios que estén conectados en la sala de edición puedan observar el cambio en la melodía. En caso de que la canción contenga muchas notas, la lista debe poder ser *scrollable*, para poder observarlas todas.

A su vez, cada nota debe seguir el siguiente formato en la lista:

nota octava intensidad duración[.]

El punto (.) es opcional. Sólo se agrega si la duración de la nota es con puntillo. Revisar la figura 4 donde hay notas que tienen punto final y otras no.

Si el cliente hace clic en alguna nota de esa lista, dicha nota es eliminada de la canción.

Además, en esa lista puede haber silencios, el formato de dichos silencios queda a su criterio. **no olviden especificar en el Readme dicho formato, aunque sea intuitivo.**

5.2.3. Sección conectados

En este **widget** deben indicar todos los usuarios que están actualmente conectados en dicha sala. Si está solo el editor, se deberá ver únicamente el nombre del usuario que está editando. Este **widget** se debe actualizar en tiempo real para saber cuando llega un nuevo usuario a la sala como espectador.

5.3. Evaluar Interfaz (Importante)

En esta tarea, toda la interacción del cliente será evaluada desde la interfaz gráfica, es decir, toda funcionalidad del *back-end* debe verse reflejada en la interfaz. **No se revisará el código para evaluar las funcionalidades.** Tampoco se evaluará la calidad de dicha interfaz (bonita, fea, escalable, etcétera), solo que posea los *input* y *widget* necesarios para demostrar el funcionamiento de su tarea.

⁶Revisar https://es.wikipedia.org/wiki/Sensible_a_may%C3%BAsculas_y_min%C3%BAsculas

6. *Bonus*

Para hacer más entretenida y amigable nuestra tarea, hemos decidido proponerles dos *bonus*. Sólo se pueden acceder a ellos si la nota sin *bonus* es mayor o igual a 4 y sólo se evaluarán mediante la interfaz. No se revisará el código en caso de no funcionar correctamente con su interfaz.

6.1. Notas simultáneas (5 décimas)

En este *bonus*, deberás estudiar y experimentar con el formato MIDI para permitir que se puedan tocar dos notas de forma simultánea. Deberás editar tu interfaz gráfica para que pueda soportar la edición de cada canal.

6.2. *Chat* (5 décimas)

Debido a que muchos usuarios han solicitado a los administradores de la interfaz que se pueda conversar con los demás usuarios dentro de una sala, se ha creado un chat.

Cada mensaje del chat debe contener el nombre de usuario que comentó, la fecha y hora en que se envió el mensaje. Al enviar un mensaje, el servidor debe procesar este mensaje para que todos los usuarios que estén conectados o se conecten en un futuro a esa sala de edición lo puedan ver.

Al ingresar a la sección de *chat*, se muestran todos los mensajes que se han escrito anteriormente en la sala. En caso de haber una gran cantidad de mensajes, su interfaz debe ser capaz de realizar un *scroll* para poder verlos todos.

7. Anexo: PEP8

En todas las tareas, se ha solicitado que su código debe seguir la guía de estilos descrita en el PEP8. A continuación se exponen los errores de PEP8 más usuales para que los tengan en cuenta para esta tarea:

1. Variables poco explicativas. Ejemplo: `a`, `var_1`, `variable`, `aux`, `concepto_no_relacionado`.
2. Uso de `CamelCase` cuando no corresponde.
3. Uso de `snake_case` cuando no corresponde.
4. No hay espacios después de las comas “,”.
5. No se respeta el máximo de caracteres por línea (79 máximo).

Los anteriores errores son los más usuales, pero no olvide que la guía posee varias reglas más que se deben respetar. Además, para esta tarea se hará un cambio en el punto 5 y se aceptará un máximo de 99 caracteres por línea, **ni un carácter más**.

8. Restricciones y alcances

- Esta tarea es **estrictamente individual**, y está regida por el *Código de honor de la Escuela*: haz clic aquí para leer.
- Si uno o más *items* de la tarea no cumplen con lo expresado en el enunciado o *issues*, dichos *items* no será evaluado.
- Tu programa debe ser desarrollado en Python v3.6.

- Tu código debe seguir la guía de estilos descrita en el PEP8.
- Si no se encuentra especificado en el enunciado, asume que el uso de cualquier librería Python está prohibida. Pregunta en el foro si es posible utilizar alguna librería en particular.
- El ayudante puede castigar el puntaje⁷ de tu tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debes adjuntar un archivo `README.md` donde comentes sus alcances y el funcionamiento del sistema (*i.e.* manual de usuario) de forma *concisa y clara*. **Tendrás hasta 24 horas después de la fecha de entrega** de la tarea para subir el `README.md` a tu repositorio.
- Crea un módulo para cada conjunto de clases. Divídelas por las relaciones y los tipos que poseen en común. **Se descontará hasta un punto si se entrega la tarea en un solo módulo**⁸.
- 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).

⁷Hasta 5 décimas.

⁸No agarres tu código de un solo módulo para dividirlo en dos; separa su código de forma lógica