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

IIC2233 Programación Avanzada (II/2017)

Tarea 3

1. Objetivos

Principales:

- Aplicar conceptos y nociones de *programación funcional* para el correcto modelamiento de un problema:
  - Uso de `map, lambda, filter, reduce`, etc.
  - Uso de generadores y funciones generadoras.
  - Uso de diccionarios, listas y conjuntos por comprensión.
- Aplicar conocimientos de *testing*.
- Aplicar conocimientos de *excepciones*.

Complementarios:

- Comprender el uso y aplicación del archivo `.gitignore`.
- Familiarizarse con `PyQt5`.
- Familiarizarse con la visualización de información.

2. Introducción

Era la batalla final. A un lado se encontraba la malvada hechicera Chau y sus caballeros, y al otro, la Reina Barrios y Lucas con sus mascotas. La cifra era dispar, el escuadrón enemigo era mayor, si se sacaban los números claramente el reino caería, y la Era del Mal comenzaría. El combate era innminente, pero antes la soberana avanzó unos pasos, miró a su compañía y dijo "¡Muchachos: la contienda es desigual, pero ánimo y valor. Nunca se ha arriado nuestra bandera ante el enemigo y espero que no sea ésta la ocasión de hacerlo!". Luego de eso, levantó su espada y corrió por los pastizales de las piezas tipo GGGGGG, y Lucas sobre sus perros la siguieron. El choque era fatal, adiós DCCesteros, adiós Reina Barrios, adiós mundo. Pero... ¡Alto! Un Ovni apareció en el centro de la pelea, abdució a nuestra monarca y se retiró de lo que antes denominábamos hogar. Dentro de la nave está ainav-I, la alienígena del planeta XOXO.

Esta extraterrestre ha estado trabajando en paralelo con Fer_and.ino, *terminator* con la cualidad de adaptarse a cualquier lenguaje al no tener acento. Este robot logró recopilar muchos genomas humanos con ciertas características específicas como color de piel, color de ojos, etcétera, para que, en caso de catásfrofes como ésta, la raza humana pueda ser recreada y así prevalecer.

Además, la nave cuenta con un tercer integrante especializado en traducción *genoma-humánico* —**TÚ**— que deberá obtener estos rasgos para que así se puedan crear humanoides en una nueva Tierra. Una vez concebidos, deberá ver quiénes están relacionados entre sí para identificar quiénes son familia y que toda la sociedad pueda desarrollarse.

3. HUMANGI

Deberás crear el programa `HUMANGI` que tendrá que leer e interpretar las secciones de código genético de una persona de tal forma en que se logren obtener sus características. Una vez que tengas la información de cada persona, tendrás que ser capaz de calcular estadísticas sobre la población como mediana de color de ojos, asimetría de altura, media de calzado de pies, promedio de orígenes, etcétera. Finalmente, deberás calcular el grado de parentesco entre personas.

Para cumplir este objetivo, deberás desarrollar una librería con todas las funciones que se explican en los siguientes ítems. También deberás generar un sistema de excepciones y testearlas para que tu programa no se caiga durante la ejecución al ingresarles ciertos comandos (ver sección 4). Luego, deberás modificar el archivo `main.py` adjunto a este enunciado, para que la interfaz otorgada sea capaz de procesar diferentes consultas sobre las personas y detectar excepciones que puedan levantarse. Todos estos requerimientos se explican más en detalle a continuación.

3.1. Glosario

- **Genotipo**: Se refiere al conjunto de genes que un organismo tiene. Es toda la información genética que se encuentra en su ADN.
- **Fenotipo**: Se refiere a las características observables de una persona debido al conjunto de genes en el ADN y al ambiente que lo afecte.

Por ejemplo, una persona puede tener en su ADN genes que indican que sus ojos son de color azul y verde, pero al final a esa persona se le observan ojos de color azul. Bajo este ejemplo, uno puede decir que el genotipo del color de ojos de esa persona es azul y verde pero su fenotipo es azul.

3.2. Lectura de Genoma [20 %]

Cada fila de `ADN` se puede separar en dos secciones, el *header* y la *data*. En el *header* se define el nombre, apellido y cómo distribuyen los genes de cada característica dentro de la sección *data*. La otra sección corresponde al conjunto de genes con los posibles valores para cada característica.

El formato del *header* es el siguiente:

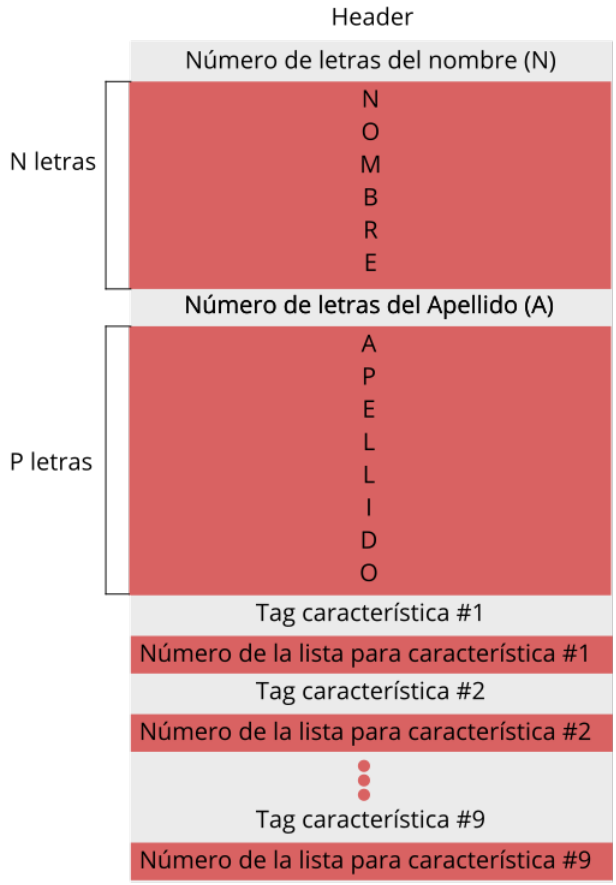
- El primer dato es un número con la cantidad de letras que tiene el nombre. El número puede ser de 1 o 2 dígitos.
- Los siguientes X elementos corresponden al nombre de la persona. X está definido según el primer dato.
- El siguiente dato es otro número con la cantidad de letras del apellido. El número puede ser de 1 o 2 dígitos
- Los siguientes "Y" elementos corresponden al apellido de la persona. "Y" está definido según el elemento anterior.

- Los siguientes 18 datos corresponden a la forma en que está codificada cada característica. Esta parte se lee de a dos datos. El primer dato es el identificador de la característica (ver Tabla 1). El segundo dato es un número que representa el *id* de una lista con las posiciones de los posibles valores para dicha característica. Para esto se entregará un archivo llamado `listas.txt` donde cada fila tendrá un *id* y una lista de valores. El *id* y la lista de valores están separados por un punto y coma (`;`), mientras que los valores dentro de la lista están separados por una coma (`,`). Un ejemplo de este archivo es:

  ```
  1; 1,2,6,8,12,13,16,35,56,123,3467, ... , 7768
  2; 3,4,5,7,8,12,13,16,23,26,27, ... , 12346
  .
  .
  .
  999; 56,89,124,236,368,447,789,1114, ... , 123456
  ```

  Para saber qué lista utilizar, deben acceder al archivo `listas.txt` y buscar aquella que tenga el mismo *id*. En esa fila está la información necesaria para encontrar los genes. Puede suponer que el *id* es único por lista.

De forma gráfica, el *header* (visto en forma vertical) tendrá el siguiente formato:



El *genoma* de cada persona contiene información de todas las características que la componen, pero en `HUMANGI` sólo se analizarán las 9 previamente mencionadas. Es por esto que hay una gran cantidad de información del *genoma* que **no deberás tomar en cuenta**. Las listas mencionadas anteriormente te guiarán para llegar a los genes deseados.

3.3. Genoma

El archivo `genoma.txt` tiene **todas** las personas, cada una en una fila. Un ejemplo de un archivo `genoma.txt` sería:

```
... ACTGAGCTAGCATGACACTACGACTCGATATCGCTATACGCATACGATCGACTAGCTAGCTA ... ATCG
... GCATACGACTACGACTACAGACTATCTCAATATCCTAGAGCGCGATATCAGACTACGACTAC ... GCTA
.
.
.
... CGTACACTGACAGCATCTGATGCATCATCAGCTACGATCAGCTACGACTACGACTACGACTG ... CGAA
```

Las características de cada persona se encuentran en su genoma (o ADN). Cada característica posee un *tag* identificador y un conjunto de *tags* de valor (cada una de 3 letras). Existen tres tipos de características: **deterministas, continuas** o **complementarias**. Cada una tiene su propia forma de definir el fenotipo a partir del genotipo.

En el **Cuadro 1** se muestran las características existentes en `HUMANGI`.

***Tag* identificador:**	**Característica:**
`AAG`	Altura
`GTC`	Color de ojos
`GGA`	Color de pelo
`TCT`	Tono de piel
`GTA`	Forma de la nariz
`CTC`	Tamaño de los pies
`CGA`	Donde hay vello corporal
`TGG`	Tamaño de la guata
`TAG`	Problemas de Visión

Cuadro 1: *Tag* identificador y característica asociada.

A continuación se explicará cómo determinar el fenotipo de cada característica dependiendo de si es **determinista**, **continua** o **complementaria**.

3.3.1. Forma Determinista

Estos genes tienen una jerarquía de dominancia. El fenotipo de una característica depende de su posición en la jerarquía y la aparición de ésta en el genoma.

La jerarquía de dominancia (de más a menos dominante) para las características deterministas es la siguiente:

- **Ojos**: Cafés ⇒ Azules ⇒ Verdes
- **Pelo**: Negro ⇒ Rubio ⇒ Pelirrojo
- **Nariz**: Aguileña ⇒ Respingada ⇒ Recta

A modo de ejemplo, si dentro de una secuencia de características de color de ojos hay genes cuyo valor son cafés y otros son verdes, el fenotipo será café.

En el **Cuadro 2** se muestra la característica y *tag* de valor correspondiente:

Ojos:			**Pelo:**			**Nariz:**		
Cafés:	**Azules:**	**Verdes:**	**Negro:**	**Rubio:**	**Pelirrojo:**	**Aguileña:**	**Respingada:**	**Recta:**
`CCT`	`AAT`	`CAG`	`GTG`	`AAT`	`CCT`	`TCG`	`CAG`	`TAC`

Cuadro 2: *Tag* de valor para el color de ojos, color de pelo y nariz.

3.3.2. Forma Continua

A diferencia de las características deterministas, el valor del fenotipo continuo está determinado por la proporción en que se encuentra un valor del gen respecto de su valor opuesto. Por ejemplo, la piel será más oscura o más clara dependiendo de la cantidad de genes blanco y moreno presentes en el ADN. Tendrás que encontrar estos valores siguiendo una relación lineal entre la proporción y el valor según los siguientes valores borde y categorías:

Característica	**Valor Mínimo**	**Valor Máximo**
Altura	1.40 (m)	2.10 (m)
Pies	Talla 34	Talla 48

Cuadro 3: Rango mínimo y máximo para la altura y tamaño de pies.

Rasgo	**Valor Mínimo**	**Intermedio 1**	**Intermedio 2**	**Valor Máximo**
Piel	Albino	Blanco	Moreno	Negro
Guata	Modelo	Atleta	Mañana empiezo la dieta	Guatón Parrillero
Intervalos	Gen 1: 100 % a 75 %	Gen 1: 74,999 % a 50 %	Gen 1: 49,999 % a 25 %	Gen 1: 24,999 % a 0 %

Dígase Gen 1 al gen ACT para la "guata", y al gen AAT para la piel. Notar que los porcentajes son respecto únicamente al gen complementario. No se deben tomar en cuenta otros genes presentes en la secuencia del carácter.

Cuadro 4: Rango mínimo, intermedios y máximo para el color de piel y tamaño de "guata".

En el **Cuadro 5** se encuentran las características y los *tags* de valor asociados:

Altura:		**Pies:**		**Piel:**		**Guata:**	
Alta:	**Baja:**	**Grandes:**	**Chicos:**	**Clara:**	**Oscura:**	**Grande:**	**Chica:**
`AGT`	`ACT`	`GTA`	`CCA`	`AAT`	`GCG`	`AGT`	`ACT`

Cuadro 5: *Tag* de valor para la altura, tamaño de pies, color de piel y tamaño de "guata".

De este modo, si luego de recolectar los genes correspondientes a la altura, una pesona posee una proporción de 75 % del gen `AGT` versus un 25 % del gen `ACT`, su altura será de 1,925 mts. En cambio, si se buscan los genes correspondientes al color de piel, te encuentras con una persona con proporción de genes `AAT:GCG` de 1:2, estás en presencia de un *morenazo*.

3.3.3. Forma complementaria

Las características complementarias son aquellas que pueden tener múltiples valores para una misma característica. Por ejemplo, "problemas de visión" y "vello corporal". El fenotipo de un gen se expresará si

y sólo si, al menos un 20 % de los genes de esa característica tiene ese valor.

En el **Cuadro 6** se encuentran las características y *tags* de valor asociados:

Vello Corporal:			**Visión:**	
Pecho:	**Axila:**	**Espalda:**	**Daltonismo:**	**Miopía:**
TGC	GTG	CCT	TTC	ATT

Cuadro 6: *Tag* de valor para el vello corporal y enfermedad asociada a la visión.

Por ejemplo, si en una secuencia de característica de "dónde hay vello corporal", tiene 50 % presencia de *pecho*, 40 % presencia de *axila* y 10 % presencia de *espalda*, la persona tendrá como fenotipo: vello corporal en el *pecho* y en la *axila*.

3.4. Parentesco

Dentro de `HUMANGI` se pueden identificar diferentes grados de parentesco entre una persona y otra, lo que determina la cercanía genética entre ambos. El parentesco puede ser de grado -1, 0, 1, 2 ó n[1]. Para determinar el vínculo genético entre dos personas, se deben cumplir todas las condiciones propuestas a continuación:

1. **Pariente grado -1**: Ambas personas tienen un fenotipo completamente distinto, es decir, ninguna de las 9 características son exactamente iguales entre las dos personas.

2. **Pariente grado 0**: Ambas personas tienen un fenotipo exactamente igual, es decir, las 9 características son exactamente iguales.

3. **Pariente grado 1**:

Característica	**Condición**
Altura	La diferencia de altura entre ambos es $\leq$ 20 cm.
Color de ojos	Tienen el mismo color de ojos.
Color de pelo	Tienen el mismo color de pelo.
Tono de piel	Tienen el mismo tono de piel.
Forma de la nariz	Tienen la misma forma de la nariz.
Tamaño de los pies	La diferencia entre la talla de sus pies debe ser $\leq$ 2 tallas.
Problemas de visión	Tienen el mismo problema de visión.

Cuadro 7: Características que definen parientes de grado 1.

4. **Pariente grado 2**:

Característica	**Condición**
Altura	La diferencia de altura entre ambos es $\leq$ 50 cm.
Color de pelo	Tienen el mismo color de pelo.
Tono de piel	Tienen el mismo tono de piel.
Tamaño de los pies	La diferencia entre la talla de sus pies debe ser $\leq$ 4 tallas.
Problemas de visión	Tienen el mismo problema de visión.

Cuadro 8: Características que definen parientes de grado 2.

5. **Pariente grado n**:

[1]Implica que existe algún parentesco pero no es posible definir con los datos actuales si es 3, 4, 5, 6, etcétera.

Característica	Condición
Altura	La diferencia de altura entre ambos es ≤ 70 cm.
Tono de piel	Tienen el mismo tono de piel.
Tamaño de los pies	La diferencia entre la talla de sus pies debe ser ≤ 6 tallas.
Tamaño de la guata	Tienen una diferencia de tamaño ≤ 1, por ejemplo "Modelo"-"Atleta".

Cuadro 9: Características que definen parientes de grado n.

3.5. Consultas [47 %]

Su programa debe ser capaz de interpretar cada secuencia de ADN tal como se describió anteriormente y debe soportar las siguientes consultas:

1. `ascendencia(persona)`: En el antiguo mundo, existían cuatro ascendencias: *mediterránea, africana, estadounidense* y *albina*. Una de las cosas que te piden es que `HUMANGI` sea capaz de identificar la ascendencia de una persona.

   La mayoría de las ascendencias se pueden identificar mediante los fenotipos, a excepción de la ascendencia *albina* la cual se podrá identificar mediante el genotipo. En la siguiente tabla se encuentran todas las características que debe tener una persona para tener cierta ascendencia.

Ascendencia	Característica 1	Característica 3	Característica 3
`Mediterránea`	Pelo Negro	Pelo en el pecho	Nariz Recta
`Africana`	Piel Negra	Pelo Negro	"Pies Grandes"
`Estadounidense`	Guaton Parrillero	Pelo en la espalda	-

Cuadro 10: Características de las ascendencias *mediterránea*, *africana* y *estadounidense*

   De esta forma, toda persona que tenga piel negra, pelo negro y "pies grandes", tendrá ascendencia *africana*. Notar que **una persona puede tener más de una ascendencia**.

   Para la ascendencia *albina*, solo deberás ver si el gen albino, `AAT`, está presente en las características donde se puede expresar el gen. Entonces, una persona será albina si el gen `AAT` está presente en sus secuencias genéticas de los **ojos, el pelo y la piel**.

2. `índice_de_tamaño(persona)`: Para realizar esta consulta, primero deberás buscar el gen de tamaño `AGT` en las características donde se expresa: la altura y el tamaño de los pies. Luego, para obtener el resultado, hay que calcular el porcentaje de aparición del gen respecto a la suma de las apariciones de él y su gen opuesto, el que achica la característica (`ACT`). Finalmente, se debe calcular la raíz cuadrada de la multiplicación de los porcentajes. Por ejemplo, si en la altura se encuentra la misma cantidad de gen `AGT` que de `ACT`, y en la guata un 20 % de los **genes que se expresan** (tomando en cuenta solo `ACT` y `AGT`) son `ACT`, el índice será: $\sqrt{0{,}5 * 0{,}8} = \sqrt{0{,}4} = 0{,}63$.

3. `pariente_de(grado, persona)`: Dado un grado (-1, 0, 1, 2 o n) y el nombre de una persona, esta consulta debe retornar el nombre de todos los parientes según el grado ingresado de la persona.

4. `gemelo_genético(persona)`: Dado el nombre y apellido de la persona, se debe encontrar la persona más parecida a ésta. Esto debe ser genéticamente y no fenotípicamente; es decir, puede que una persona de pelo negro sea más parecida a una persona rubio en vez de una persona de pelo negro, por la cantidad de genes de ese tipo que tenga. Una persona se declara parecida a otra según la cantidad de genes en común que tengan para cada característica.

5. `valor_característica(tag_identificador, persona)`: Dado el *tag* identificador de una característica, la consulta debe retornar el valor de ésta en la persona. Por ejemplo, si se quiere consultar el color de ojos de "Hernán Valdivieso", `valor_característica(GTC, ‘‘Hernán Valdivieso")` debe retornar el valor "Café".

6. Consultas estadísticas:

   a) `min(tag_caracteristica)`:
   - Si el *tag* identificador corresponde al color de ojos, color de pelo, tono de piel o forma de nariz, entonces, la consulta debe retornar el valor menos frecuente fenotípicamente. Por ejemplo `min(GGA)`, retornará "Pelirrojo" si este color de pelo es el menos usual.
   - Si el *tag* identificador corresponde a la altura o tamaño de los pies, la consulta debe retornar el menor valor encontrado fenotípicamente. Por ejemplo `min(CTC)`, retornará "35" si esta talla de pie es la más pequeña registrada.

   b) `max(tag_caracteristica)`:
   - Si el *tag* identificador corresponde al color de ojos, color de pelo, tono de piel o forma de nariz, entonces, la consulta debe retornar el valor más frecuente fenotípicamente. Por ejemplo `max(GGA)`, retornará "Negro" si este color de pelo es el más usual.
   - Si el *tag* identificador corresponde a la altura o tamaño de los pies, la consulta debe retornar el mayor valor encontrado fenotípicamente. Por ejemplo `max(CTC)`, retornará "43" si esta talla de pie es la más grande registrada.

   c) `prom(tag_caracteristica)`:
   - Si el *tag* identificador corresponde a la altura o tamaño de los pies, la consulta debe retornar el valor promedio de los datos registrados fenotípicamente. Por ejemplo `prom(AAG)`, retornará "1.71" si es el promedio de las alturas revisadas. En otro caso, no se podría procesar la consulta.

3.6. Formato de consultas

El formato de estas será una lista cuyo primer elemento es el nombre de la consulta y los siguientes K elementos corresponden a los argumentos que requiere la consulta. Es decir:

```
["nombre", "arg_1", "arg_2", ... ,"arg_n"]
```

El programa a realizar debe soportar el ingreso de un conjunto de consultas. Por lo tanto, el programa recibirá una lista de consultas en la forma:

```
[[consulta_1], [consulta_2], ... ,[consulta_K]]
```

Ejemplos de *input* serían:

```
una_consulta = [["pariente_de", 0, "Tanya Garrido"]]
dos_consultas = [["pariente_de", 0, "Hernán Valdivieso"],["gemelo_genético", "Fernando Pieressa"]]
```

3.7. Bonus: *Bubble charts* [6 décimas]

Muchas veces el **rico y mafioso Nebil** querrá visualizar la información de la gente y lamentablemente, no tiene tiempo para crear interactivos gráficos en `D3.js`. Afortunadamente, se conformó con aceptar gráficos estáticos creados con `matplotlib`[2] y darles un bonus a todos los alumnos que logren hacer dos *bubble charts*.

[2]Si tienes Ubuntu y no te funciona bien `matplotlib`, dando error de `tkinter`, debes ejecutar el siguiente comando en la terminal: `sudo apt-get install python3-tk`

Estos gráficos deben considerar a todas las personas en la base de datos, por lo que se deberá graficar con respecto a todos los genes existentes. Se debe poder elegir dos gráficos distintos, uno que sea con respecto a su color de ojos y otro con respecto a su color de pelo. Para los gráficos deberás utilizar distintos colores donde las burbujas deben representar los posibles colores de la característica que se haya elegido (pelo u ojos). Por ejemplo, si se elige el color de ojos, se deberían mostrar burbujas azules, verdes y cafés. Además del color de burbujas, los gráficos debe incluir lo siguiente:

- El **área** de la burbuja es proporcional a la cantidad de gente que poseen el color de ojos o de pelo en común.
- La posición Y de cada burbuja será el promedio de las alturas de todas las personas que tengan ese mismo color de ojos o pelo.
- La posición X de cada burbuja será el promedio del tamaño de los pies de todas las personas que tengan ese mismo color de ojos o pelo.

La forma de realizar el gráfico será mediante una consulta del modo: `visualizar(tipo)` donde tipo puede ser "ojos" o "pelo".

4. Excepciones [13 %]

Se espera que se manejen todos los errores posibles mediante el uso correcto de excepciones. **Una excepción genérica**[3] **no se considerará correcta**, a menos que sea la única solución posible y deberás explicar en el `README.md` por qué es la única solución posible.

Dado que estarás trabajando con genomas "humanos", deberás crear nuevas excepciones las cuales están detalladas a continuación:

- `Bad Request:`
  Esta excepción se genera cada vez que se ingresa una consulta mal escrita o que no existe dentro de las consultas solicitadas y no se puede procesar.
- `Not Found:`
  Se debe levantar esta excepción si se ingresan mal los parámetros de una consulta.
- `Not Acceptable:`
  Se debe levantar esta excepción si el resultado de la consulta es vacío.
- `Genome Error:`
  Se debe levantar esta excepción si al leer un genoma, éste tiene errores ya sea porque tiene espacios, letras distintas a `ACTG`, números o caracteres especiales.

5. *Testing* [20 %]

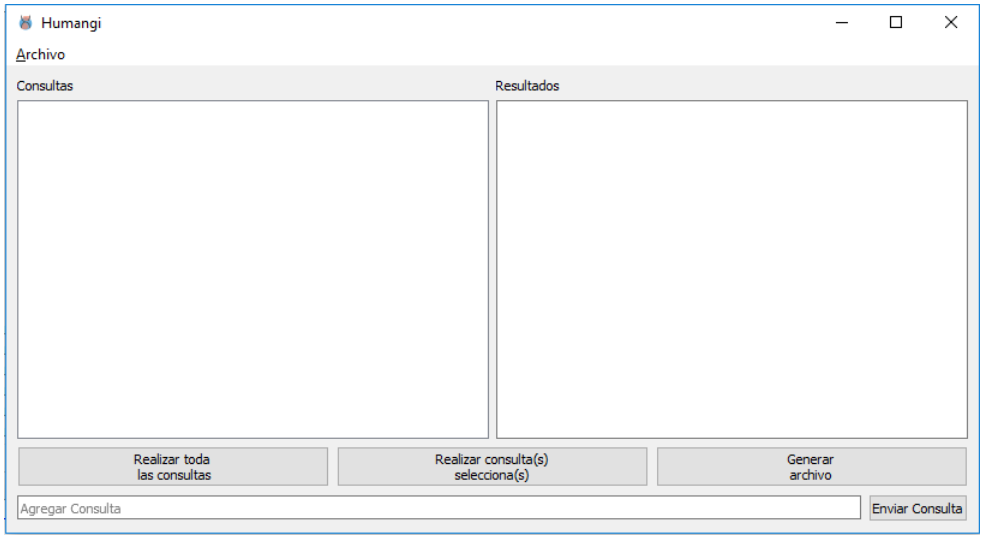
En un módulo llamado `testing.py` deberás testear **todas** las excepciones indicadas en la sección 4 y **todas** las consultas. Para testear las consultas, todas deben probarse con un caso y en dos de ellas probar dos casos de uso distintos. Para realizar los *tests*, debes utilizar la librería de Python `unittest` sobreescribiendo los métodos de `setUp` y `tearDown`.

[3] `except Exception`

6. GUI

Para desarrollar el programa, se te entregará una archivo llamado `main.py` con una interfaz el cual debes utilizar para analizar las consultas. Éstas se pueden ingresar directamente utilizando la interfaz o se pueden cargar desde el archivo `consultas.json`. Las respuestas deberán escribirse en el un archivo llamado `resultados.txt` y se podrá mostrar en la interfaz en paralelo con la consulta asociada cuando se solicite. **No debes modificar la interfaz**; sólo puedes interactuar con ella a través de los métodos que se especifican más adelante.

La interfaz que se les entrega es la siguiente:



Se te entregará el archivo `main.py` que contiene los métodos que deben modificar para trabajar con la interfaz gráfica.

6.1. Métodos de la GUI

- `add_answer(text)`: Este método se encarga de agregar cualquier texto al cuadro **Resultados** de la interfaz. Debe recibir un ***string*** de argumentos y **deben** incluir los saltos de línea (`\n`) si quieren poner líneas por separado.

- `process_query(query_array)`: Se llama cuando se presiona el botón "Realizar todas las consultas" o "Realizar consulta(s) seleccionada(s)". Recibe un parámetro llamado `query_array`, que es una *lista* con todas las consultas seleccionadas en la interfaz (o todas, en caso de apretar el botón "Realizar todas las consultas").
  Debes modificar este método llamando a tu librería para procesar las consultas y con el método `add_answer` puedes mostrar los resultados de las consultas procesadas en el cuadrado **Resultados** de la interfaz.

- `save_file(query_array)`: Se llama cuando se presiona el botón "Generar archivo". Recibe un parámetro llamado `query_array`, que corresponde a una lista que contiene a todas las consultas que se han ingresado a la interfaz (es la misma lista que se recibiría en `process_query` si se oprime "Realizar todas las consultas").
  En este método debes implementar todo lo necesario procesar y escribir los resultados en `resultados.txt` respetando el formato entregado.

7. Archivos relacionados a la tarea

¡Esta sección es muy importante! No respetar el formato afectará considerablemente la nota de su tarea.

7.1. `consultas.json`

Este archivo tendrá el siguiente formato (donde `k` es la cantidad de consultas):

```
[
  [consulta 1],
  [consulta 2],
  ...                ,
  [consulta k]
]
```

Para poder leer este archivo, se debe copiar, **sin cambiar su nombre**, a la misma carpeta que `main.py`. A través de la interfaz, en la barra de opciones, debe seleccionar la opción "Abrir" para cargar el archivo.

7.2. `resultados.txt`

Este archivo debe indicar como mínimo el numero de consulta y su resultado, es decir, debe tener el siguiente formato:

```
------------ Consulta 1 -----------
RESULTADO CONSULTA 1
------------ Consulta 2 -----------
RESULTADO CONSULTA 2
...
----------- Consulta K -----------.
RESULTADO CONSULTA K
```

7.3. Consultas fallidas, ejecución y ejemplo

Su programa debe:

1. Leer `consultas.json`
2. Generar `resultados.txt` y **no caerse** al encontrar un error durante el procesamiento de alguna consulta, sino indicar el error con el formato que se explicó previamente. La idea es que pueda seguir con las otras. Un ejemplo de manejo de errores a continuación:

```python
for i in ["1", 2, 4, "3", 5, 6, "4"]:
    try:
        print(i + 1)
    except TypeError:
        print("Error")
# Se imprime lo siguiente
Error
3
5
Error
6
7
Error
[Finished in 0.1s]
```

Un ejemplo[4] de la lógica de `main.py` a continuación:

```
if __name__ == "__main__":
    # Abre consultas.json
    for i in consultas:
        try:
            resultado = procesar_consulta(i)
            # Escribe los resultados en resultados.txt
        except MyExcepcion:
            imprimir_fallo()
            # Escribe que la consulta falló en resultados.txt e indica el nombre del error
```

De esta forma, el archivo `resultados.txt` debería verse como:

```
-------- CONSULTA 1 ----------
FALTA RESPUESTA #tipo_de_error
------- CONSULTA 2 ----------
FALTA RESPUESTA #tipo_de_error
------- CONSULTA 3 ----------
FALTA RESPUESTA #tipo_de_error
------- CONSULTA 4 ----------
FALTA RESPUESTA #tipo_de_error
```

8. Consideraciones

8.1. Programación funcional

`HUMANGI` debe estar implementado usando programación funcional. Se permite el uso de *loops* (`for` y `while`) **solo en funciones generadoras, expresiones generadoras, o bien, en contenedores (e.g. listas, diccionarios, tuplas) por comprensión**. A continuación, se mostrarán ejemplos del uso de *loops* permitidos y no permitidos[5]:

```
# Este for no se puede utilizar
for gen in genes:
        separar_gen(gen)

# En cambio, este for si se puede utilizar
primer_elemento = (x[0] for x in listas_de_listas)

# Recuerde que debe utilizar programación funcional para toda la implementación de
# inserte_nombre,como por ejemplo funciones lambda y map:
dobles = map(lambda x: x*2, numeros)
```

Si está permitido utilizar `for` y `while` para acceder a cada consulta dentro de la lista de consultas, imprimir los resultados en pantalla y escribir los resultados en el archivo.

```
# Este for si se puede utilizar
for query in array_query:
    result = process_query(query)
    self.add_answer(result)

# Esto otro tambien se podría
results = process_queries(array_query)
with open("resultados.txt", "w") as file_:
    for result in results:
        file_.write(result)
```

[4] Es solo un ejemplo: pueden escribirlo de la forma que quieran, siempre que se cumpla el procedimiento pedido y el formato mínimo.

[5] Nos interesa que no usen los ciclos `for` y `while` en lo que son las funcionalidades lógicas del programa. Sí pueden utilizarlos, por ejemplo, para imprimir valores.

8.2. Funciones atomizadas

Dada la similitud en la manera que se deben realizar ciertas búsquedas, es un requisito que su librería implemente las funciones de forma atomizada. Esto significa que deben haber funciones pequeñas, que hagan algo específico y corto, que pueda ser generalizado y usado otras partes del programa. Debido a esto, se evaluará que las funciones no superen el **máximo de 15 líneas**, considerando la definición de la función (`def function(*args)`) como una de ellas.

Si consideras que una función tuya ya está atomizada, pero tiene un mayor largo de la cantidad permitida, **debes** justificar en el `README` por qué crees que está atomizada.

8.3. Uso de clases

Para esta tarea **no se permite el uso de clases** excepto para las excepciones personalizadas y el *testing* si lo considera necesario.

8.4. `.gitignore`

Para no saturar los repositorios de GitHub, **deberás** agregar en tu carpeta `Tareas/T03/` un archivo llamado `.gititgnore` de tal forma de **no** subir los archivos de la carpeta `gui`. La siguiente página les será muy útil para crear el contenido del archivo: www.gitignore.io.

9. Entregable

Deberás entregar un detalle de cómo lograrás leer el genoma de una persona y cómo implementarás las consultas `ascendencia(persona)` y `pariente_de(grado, persona)`. Para la lectura del genoma debes especificar:

1. Que *built-ins* de Python utilizarás, es decir, si usarás `map, lambda, filter`, generadores, etc., y con qué fin.
2. Indicar las funciones atomizadas necesarias para lograr el objetivo. Debes incluir el/los parámetro(s) que recibirá, qué hace la función y qué tipo de dato retornará.

- Con respecto a cómo implementar las consultas de `ascendencia(persona)` y `pariente_de(grado,persona)`, deben especificar:
    1. Todas las funciones atomizadas que crearás para cumplir esa consulta. En cada función, debes incluir los parámetros que recibe, qué hace la función y qué tipo de dato retornará.
    2. Describir, paso a paso, cómo la consulta irá llamando a las funciones para lograr el objetivo.

10. Restricciones y alcances

- Tu programa debe ser desarrollado en Python v3.6.
- Esta tarea es estrictamente individual, y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- 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 de Python está prohibida. Pregunta en el foro si es que es posible utilizar alguna librería en particular.

- El ayudante puede castigar el puntaje[6] 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.
- El no uso del archivo `.gitignore` tendrá un fuerte descuento.
- 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[7].**
- Se descontarán 3 décimas sin no se entrega el archivo preliminar (entregable).
- **No se corregirán tareas que no interactúen con la GUI.**
- Cualquier aspecto no especificado queda a tu criterio, siempre que no pase por sobre otro.
- **No se otorgará puntaje a los ítems que no se implementen con la materia de funcional. En otras palabras, el uso de ciclos prohibidos y clases (que no se usen para las excepciones personalizadas).**

11. Entrega

- `Entregable`
  - **Fecha/hora:** 27 de septiembre del 2017, 23:59 horas.
  - **Lugar:** Cuestionario Siding
- `Tarea`
  - **Fecha/hora:** 8 de octubre del 2017, 23:59 horas.
  - **Lugar:** GitHub – Carpeta: `Tareas/T03/`
- `README.md`
  - **Fecha/hora:** 9 de octubre del 2017, 23:59 horas.
  - **Lugar:** GitHub – Carpeta: `Tareas/T03/`

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).

[6] Hasta −5 décimas.
[7] No agarres tu código de un solo módulo para dividirlo en dos. Separa tu código de forma lógica