TALLER PROGRAMACIÓN ORIENTADA A LA BIOLOGÍA

(En el marco del PROYECTO DE EXTENSIÓN DE BIOINFORMÁTICA EN EL AULA)

Guía de ejercicios complementarios

La bioinformática es la aplicación de métodos computacionales al análisis de datos biológicos para poder contestar numerosas preguntas. Las tecnologías computacionales permiten, entre otras cosas, el análisis en plazos cortos de gran cantidad de datos (provenientes de experimentos, bibliografía, bases de datos públicas, etc), así como la predicción de la forma o la función de las distintas unidades biológicas, o la simulación del comportamiento de sistemas biológicos complejos. Además de ser un campo de estudio dentro de la bioquímica y la biología, la bioinformática puede ser pensada como una herramienta en el aprendizaje de la biología: su objeto de trabajo son entidades biológicas (ADN, proteínas, organismos completos, sus poblaciones...), estudiadas con métodos que permiten 'visualizar' distintos procesos de la naturaleza. Así, la bioinformática aporta una manera clara y precisa de percibir los procesos biológicos, y al mismo tiempo acerca a los estudiantes herramientas que integran múltiples conocimientos (lógico-matemático, biológico, físico, estadístico) generando un aprendizaje significativo y envolvente.

Bienvenido!! ¿Estás listo?

Para comenzar podríamos definir algunas cosas:

¿En qué consiste una computadora?

Una computadora está formada por el *Hardware* (que son todas las partes o elementos físicos que la componen) y el *Software* (que son todas las instrucciones para el correcto funcionamiento del *Hardware*). El software operativo o sistema operativo proporciona una interfaz con el usuario y es el que permite al resto de los programas una interacción correcta con el *Hardware*.

¿Qué hacemos entonces cuando programamos?

Los lenguajes de programación actúan como traductores entre el usuario y el equipo. Un programa intermedio, denominado *Compilador*, convierte las instrucciones dadas en un determinado lenguaje al *lenguaje máquina* (un lenguaje simple, que sólo consta de ceros y unos). Una computadora está constituida, básicamente, por un gran número de circuitos eléctricos que pueden ser activados (1) o desactivados (0). Al establecer diferentes combinaciones de prendido y apagado, se logra que el equipo realice alguna acción, por ejemplo, que muestre algo en la pantalla. Eso es la programación.
En lugar de aprender el lenguaje nativo de la máquina, se puede utilizar un *lenguaje de programación* para dar instrucciones al equipo de un modo que sea más fácil de aprender y entender. Esto significa que, como programador de Python (o cualquier otro lenguaje), no necesitas entender lo que el equipo hace o cómo lo hace, basta con que entiendas cómo funciona el lenguaje de programación.

¿Por qué es útil aprender a programar?

Tu smartphone, tu Playstation o Wii no serían muy útiles sin programas (aplicaciones, juegos, etc) para hacerlas funcionar. Cada vez que abrimos un documento para hacer un trabajo práctico para la escuela o usamos el WhatsApp para chatear con nuestros amigos estamos usando programas que interpretan lo que deseamos (un color de fuente, un formato de letra, 'enviar un mensaje') y se lo comunican a nuestra PC o teléfono para que ejecute nuestra orden. Aprendiendo a programar podrías hacer una gran diversidad de cosas: desde escribir tus propios juegos, aplicaciones para celular o

combinar el uso de varios programas de forma secuencial o leer millones de texto sin abrir un solo libro, hasta analizar el genoma de un organismo o un millón de estructuras proteicas y sacar conclusiones de relevancia biológica.

Entonces: ¿Qué es Python?

Es un lenguaje de programación con una forma de escritura (sintaxis) de fácil lectura. Es lo que se conoce un lenguaje de *scripting*, que puede ser ejecutado por partes y no necesita ser compilado en un paso aparte. Python tiene muchas características, ventajas y usos que vamos a pasar por alto en este taller, pero que podes leer en las páginas oficiales de Python y Python Argentina. Para nosotros, por el momento, la razón más importante para elegir Python es que podés comenzar a hacer tus propios programas realmente rápido.

¿Cómo usamos Python?

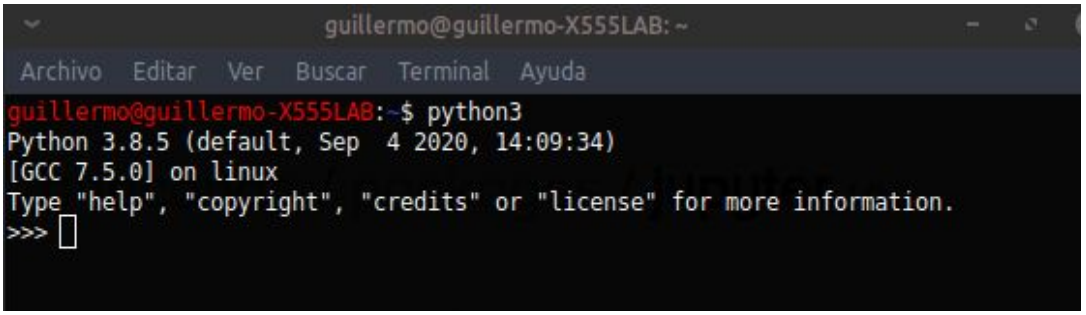
Esto depende de con qué dispositivo contemos, te dejamos una guía detallada de como instalar Python en todos los sistemas operativos y en distintos dispositivos.

"Aún el camino más largo siempre comienza con el primer paso" - Lao Tse

Un primer paso para poder hacer tu primer programa es abrir la consola de Python, tu App del teléfono o consola en línea, lo que tengas a mano para arrancar!
Si estás desde tu PC la forma es:

Linux

Abrir una terminal (si no tenés creado el acceso directo, podes tipear en el inicio 'terminal' para encontrarla) y escribir:
python3



Windows

Ir al inicio, buscar la carpeta de Anaconda. Ingresar y clickear en el ícono de Jupyter Notebook. Abrir un nuevo Notebook (New -> Python 3) y listo! Si querés trabajar desde la consola de Python sin la interfaz interactiva: buscá en el inicio Python 3.8. Hacé click en la aplicación Python Prompt y ya estás listo para empezar!

Si todo está bien, vas a ver lo que se denomina 'prompt', que en el caso de Python son tres signos mayor '>>>'. El prompt aparece siempre que Python esté listo para recibir una nueva orden.

AHORA SÍ: Manos al teclado!!

La importancia de las comillas en Python

Como vimos durante el taller, el operador matemático "+" no solo sirve para sumar dos números, sino también para concatenar palabras (*strings*).

> **RETO 1: Utilizando nada más que la función `print()`, el símbolo `+` y la cantidad necesaria de números `7` y comillas " ", hacé programas que impriman las tres líneas siguientes?**
>
> ```
> 14
> 7+7
> 77
> ```
>
> PISTA: como dice el título, las comillas (o la falta de ellas) es crucial

¡Similar, no es igual!

Como bien sabemos, Python distingue fácilmente entre dos palabras diferentes: ¡caracteres distintos, palabras distintas! Además, no solo podemos decidir si dos *strings* son iguales o no, si no que Python nos permite evaluar cuál de las dos palabras es la más "grande" mediante los operadores relacionales que ya conocemos: "`>`" y "`<`". Tomando como referencia el orden alfabético, las palabras más "grandes" serían las que aparecen más tarde en nuestro alfabeto. Python también es capaz de distinguir sutilezas no tan sutiles, como diferencias en mayúsculas y minúsculas. Palabras con iguales caracteres pueden ser distintas para Python si contienen mayúsculas, por ejemplo 'Genes' o 'genes'.

> **RETO 2: Teniendo en cuenta todo esto:**
>
> a) **Escribí un programa que indique si "a" es mayor, menor o igual que "z".**
> b) **Escribí un programa que indique si "a" es mayor, menor o igual que "A".**
> c) **Escribí un programa que indique si "a" es mayor, menor o igual que "Z".**
> d) **Escribí un programa que indique si "A" es mayor, menor o igual que "Z".**
>
> **Sobre la base de los resultados anteriores y de acuerdo a Python, ¿qué alfabeto de Python está antes, el de las minúsculas o el de las mayúsculas?**

Vayamos por partes…

Hemos visto que las listas de Python pueden "cortarse en fetas" utilizando los corchetes: por ejemplo, `lista[a:b]` extrae todos los elementos de "lista" comprendidos entre las posiciones "a" y "b-1". En el siguiente ejemplo usamos las funciones `list()` y `range()` para definir una lista que contiene los números del 0 al 9, y luego imprimimos sólo sus primeros cinco elementos:

```
lista_numeros = list(range(0,10))
print(lista_numeros[0:5])
```

> **RETO 3: Escribí un programa que genere e imprima una lista de todos los números pares entre 0 y 30 inclusive. Podés hacerlo de distintas formas: ¿cuántos programas distintos se te ocurren?**

PISTA: al "cortar en fetas", podés definir el "salto" entre una feta y otra definiendo el parámetro "c" en `list[a:b:c]`.

Te comento algo

En Python tenemos la posibilidad de incluir texto que, aunque esté escrito en el programa, no deba ejecutarse. Esto se logra empezando la línea con el símbolo `#` (el *hash* o numeral). Estas líneas se llaman "comentarios" y se utilizan para incluir en el programa algunas aclaraciones acerca del código útiles para nosotros, por ejemplo:

```
# definir la variable 'nombre' e imprimirla
nombre = "Ana"
print(nombre)
```

RETO 4: Sobre la base del ejemplo anterior, indicá cuál es el efecto (y qué lo causa) de agregar un símbolo # al principio de las tres líneas presentadas debajo, de a una por vez:

```
# imprime del 0 al 9
for i in range(0,10):
  print(i)
```

¡Atender, atender! ¡Que comienza la función!

¿Vos también flashaste obra de teatro? Bueno, en realidad hablamos de otro tipo de funciones… Nos referimos a esos bloques de código a los que les ponemos un nombre (¿Cuca? 🐭), que ejecuta las operaciones deseadas y devuelve un valor o realiza una tarea.

Hasta ahora hemos venido ejecutando código línea a línea para lograr nuestro cometido. Pero, si bien no es estrictamente necesario que tu código tenga funciones para hacer lo que querés que haga, sí es muy recomendable. ¿Por qué? Bueno, las funciones nos permiten separar las tareas y reutilizarlas en otros programas.

¿Cómo se usan estas funciones? ¿Cómo hago para obtener resultados? ¿Cómo puedo indicarles ciertos parámetros que modifiquen los resultados obtenidos?

Basta con poner el nombre de la función y, entre paréntesis, sus argumentos. Veamos entonces cómo es que se define una función:

```
def funcion(argumento):
   Operación sobre el argumento
   return aquí va el resultado quiero devolver
```

Así por ejemplo si quisiéramos escribir una función que nos permita obtener el doble de un número, podemos escribirla del siguiente modo:

```
def doble(numero):
  resultado = numero * 2
  return resultado
```

o lo que sería equivalente y hasta mejor:

```
def doble(numero):
  return numero * 2
```

Cómo ves definir funciones en Python resulta relativamente sencillo, estas nos permiten recibir múltiples parámetros, hacer algo con ellos y devolver un resultado procesado ¡Unas verdaderas maquinitas!

Todo muy lindo, pero dijimos "múltiples parámetros" y el ejemplo que escribimos arriba solo tomaba uno. Veamos un ejemplo un poquitín más complejo y más interesante. Supongamos que somos microbiólogos[1] que estamos cultivando *Lactobacillus*[2], unas bacterias que usaremos luego para producir yogurt ¡Pero como es bien sabido, muchas bacterias juntas pueden ser un problema! Es por eso que queremos tener una estimación del número de bacterias que estaremos agregando a nuestro yogurt (inóculo). Vamos entonces a construir una función que dado un número de bacterias inicial, nos estime cuál será el recuento de bacterias a cierto tiempo.

RETO I: Definí una función que permita calcular las células totales a un tiempo dado, a partir de un dado inóculo. Tené en cuenta que el cálculo de las células totales se corresponde con:

```
cantidad_celulas_totales = cantidad_inicial_celulas * (2** tiempo)
```

Ya sabemos cómo definir funciones y podemos reciclar código y automatizar cálculos
¡¡Vamos entonces a ponernos manos a la obra!!

Bioinformática: La vida en ceros y unos

1) Dentro de la célula, el ADN guarda la información necesaria para construir las proteínas. El ADN es una cadena formada por muchas combinaciones de cuatro nucleótidos A, C, G y T. Las distintas combinaciones le otorgan distintas funcionalidades. Para que la información llegue del ADN a las proteínas, el mensaje genético es copiado desde el ADN a otra molécula, el ARN. El ARN es muy similar al ADN pero con el nucleótido U en vez de T. Este es libre de moverse del núcleo al citosol para guiar la síntesis de proteínas.

En el ARN, cada combinación de tres nucleótidos/letras (o triplete) se denomina codón. Cada aminoácido de las proteínas está codificado por uno o varios codones. Por ejemplo, el codón 'AUG' codifica para el aminoácido Metionina , el codón 'AAA' para Lisina, el codón[3] 'CUA' para Leucina, etc. Tomemos por ejemplo la siguiente codificación[4], una versión simplificada del código genético universal[2]:

[1] Científica/o que estudia microorganismos, seres vivos diminutos no visibles al ojo humano.
[2] Bacterias habitualmente benignas, que habitan en el cuerpo humano y en el de otros animales; estando presentes, por ejemplo en el tracto gastrointestinal.
[3] Codón: La información genética transportada por el ARN consiste de repeticiones de cuatro letras (A, C, G y U), que corresponden a sus bases nitrogenadas, y están dispuestas en una determinada secuencia. Estas letras pueden agruparse funcionalmente de tres en tres, en 64 formas distintas. Cada uno de estos grupos de tres letras se llama codón. Cada codón codifica un aminoácido o un símbolo de puntuación (comienzo o parada de la construcción de la proteína correspondiente).
[4] El código genético es el conjunto de reglas que define cómo se traduce una secuencia de nucleótidos en el ARN a una secuencia de aminoácidos en una proteína. Este código define la relación entre cada secuencia de tres nucleótidos, llamada codón, y cada aminoácido; y se dice que es universal ya que el mismo triplete en diferentes especies codifica para el mismo aminoácido. La principal excepción a la universalidad es el código genético mitocondrial.

Aminoácido	3 letras	1 letra	Codón ARN
Alanina	Ala	A	GCU
Arginina	Arg	R	AGA
Asparagina	Asn	N	AAU
Aspartato	Asp	D	GAU
Cisteína	Cis	C	UGC
Fenilalanina	Phe	F	UUU
Glicina	Gly	G	GGC
Glutamato	Glu	E	GAA
Glutamina	Gln	Q	CAG
Histidina	His	H	CAU
Isoleucina	Ile	I	AUU
Leucina	Leu	L	UUA
Lisina	Lys	K	AAA
Prolina	Pro	P	CCU
Serina	Ser	S	UCU
Tirosina	Tyr	Y	UAU
Treonina	Thr	T	ACU
Triptófano	Trp	W	UGG
Valina	Val	V	GUA
Metionina/Codón de Inicio	Met/Start	M	AUG
Codón de terminación	Stop		UAA

Como una forma de "acortar" la cantidad de letras que se usan para simbolizar un aminoácido, se establecieron por convención las **nomenclaturas de 3 letras y de 1 letra**. Entonces, por ejemplo, la forma más corta de simbolizar Asparagina puede ser usando "Asn" o, más simple aún, solo "N".

a) ¿Podrías escribir un programa que imprima una cadena de ARN codificante para el siguiente péptido (cadena corta de aminoácidos). PISTA: usá bucles, condiciones y operadores.

Sec1: `'AGEKGKKIFVQKCSQCHTVCSQCHTVEKGGKHKTGPNEKGKKIFVQKCSQCHTVLHGLFGRKTGQA'`

b) ¿Podrías hacer que tu programa cuente el contenido de G y C (en términos de porcentaje, respecto al largo total de la secuencia) sobre la secuencia de ARN generada?

---

2) En muchos de los genes codificados en el ADN existe un motivo recurrente ubicado antes de la secuencia codificante del gen. Se trata de una palabra específica para la unión de ARN Polimerasa II, la proteína encargada de copiar el ADN a un ARN mensajero, y su secuencia es `'TATAAA'`. Para los siguientes fragmentos de ADN, ¿podrías identificar si poseen la región de unión a la polimerasa[5]?

Sec2:

```
'GTTATAATATTGCTAAAATTATTCAGAGTAATATTGTGGATTAAAGCCACAATAAGATTTATAAATCTTAAATG
ATGGGACTACCATCCTTACTCTCTCCATTTCAAGGCTGACGATAAGGAGACCTGCTTTGCCGAGGAGGTACTACA
GTTCTCTTCACAAACAATTGTCTTACAAAATGAATAAAACAGCACTTTGTTTTTATCTCCTGCTTTTAATATGTC
CAGTATTCATTTTTGCATGTTTGGTTAGGCTAGGGCTTAGGGATTTATATATCAAAGGAGGCTTTGTACATGTGG
GACAGGGATCTTATTTTAGATTTATATATCAAAGGAGGCTTTGTACATGTGGGACAGGGATCTTATTTTACAAAC
AATTGTCTTACAAAATGAATAAAACAGCACTTTGTTTTTATCTCCTGCTCTATTGTGCCATACTGTTGAATGTTT
ATAATGCATGTTCTGTTTCCAAATTTCATGAAATCAAAACATTAATTTATTTAAACATTTACTTGAAATGTTCAC
AAACAATTGTCTTACAAAATGAATAAAACAGCACTTTGTTTTTATCTCCTGCTTTTAATATGTCCAGTATTCATT
TTTGCATGTTTGGTTAGGCTAGGGCTTAGGGATTTATATATCAAAGGAGGCTTTGTACATGTGGGACAGGGATCT
TATTTTAGATTTATATATCAAAGGAGGCT'
```

Sec3:

```
'ACTTTGTTTTTATCTCCTGCTCTATTGTGCCATACTGTTGAATGTTTATACCCTACATGGTGCATGTTCTGTTT
CCAAATTTCATGAAATCAAAACATTAATTTATTTAAACATTTACTTGAAATGTTCACAAACAATTGTCTTACAAA
ATGAATAAAACAGCACTTTGTTTTTATCTCCTGCTTTTAATATGTCCAGTATTCATTTTTGCATGTTTGGTTAGG
CTAGGGCTTAGGGATTTATATATCAAAGGAGGCTTTGTACATGTGGGACAGGGATCTTATTTTAGATTTATATAT
CAAAGGAGGCT'
```

[5] ARN polimerasa II es una proteína de eucariotas que cataliza la transcripción del ADN a precursores de ARN mensajero, microARNs y otros tipos de ácido ribonucleico.

3) Se denomina mutación puntual a los cambios que alteran la secuencia de nucleótidos del ADN. Aún cuando involucran un solo sitio o nucleótido, las mutaciones en la secuencia del ADN (y, por consiguiente, en el ARN generado a partir de él) pueden llevar a la sustitución de aminoácidos en las proteínas resultantes. Por ejemplo, imaginemos que una mutación produce el cambio de un nucleótido 'U' por 'A' en la segunda posición de un codón codificante para Leucina ('UUA'), esto daría como resultado el cambio de Leucina por un codón de terminación ('UAA'). De este modo, una mutación puntual puede tener como resultado una proteína más corta y no funcional, si se introduce antes de lo esperado un codón de terminación.
Tomando nuevamente la codificación simplificada del ejercicio 1 y considerando las primeras tres bases como el codón de inicio, ¿podrías construir un programa que nos diga qué longitud tendrán las péptidos que se generarán a partir de las secuencias de ARN especificados a continuación, si sobre ellos se produce la mutación puntual mencionada antes para el codón de Leucina?

```
Sec4:'AAUCAUGUAGUAGGCUUUUUUUUAGAUCAUGCU'

Sec5:'GCUUAUCCUGCUUUGGCUCUGGCGUAUUAACUG'

Sec6:'GCUUUAUAUCCUGCUUUGGCUCUGGCGUAUUAA'
```

UNA PISTA...
En la guía de trabajo presentamos la función range() que nos permite recorrer un intervalo numérico pasándole los argumentos de comienzo y de finalización del intervalo:

```
range(comienzo,fin)
```

Pero range nos permite además ir recorriendo el intervalo deseado de a pasos de la longitud que deseemos, es decir de dos en dos o tres en tres:

```
range(comienzo, fin,  longitud_del_paso)
```

Así por ejemplo si hacemos:

```
range(0,10,2)
```

lo que hacemos es recorrer el intervalo de 0 a 10 de dos en dos (2, 4, 6…).

4) Existen mutaciones típicamente asociadas a ciertas proteínas. Imaginemos que trabajamos con una proteína para la que cierta mutación puntual se asocia a una enfermedad inmunológica. En dicha patología los pacientes presentan intercambiada la Histidina por una Leucina en el fragmento `'HTVCSQ'`. ¿Podrías construir un programa que identifique cuáles de los siguientes pacientes poseen la enfermedad y cuáles no? PISTA: el resultado puede estar en forma de lista.

Paciente1: `'AGEKGKKIFVQKCSQCLTVCSQCHTVEKGGKHKTGPNEKGKKIFVQKCSQCHTVLHGLFGRKTGQA'`

Paciente2: `'AGAKGKKIFVQKCSQCLTVCSQCHTVEKGGKHTVCSQEKGKKIVVQKCSQCLTVLLGLFGRKTGQA'`

Paciente3: `'AGEKGKKIFVQKCSQCLTVCSQCHTVEKGGKHKTGPNEKGKKIFVQTAKCSQCHTVLLAKTGHLQA'`

Paciente4: `'AGAKGKKIFVQKCSQCHTVCSQCHTVEKGGKHKTGPNEKGKKIVVQKCSQCLTVLLGLFGRKTGQA'`

Paciente5: `'AGAKGKKIFVQKCSQCNPLCSQCHTVEKGGKHKTGPNEKGKKIVVQKCSQDETVLLGLFGRKTGQA'`

5) Como resultado de la evolución biológica, y dado que todos los organismos provienen de ancestros comunes, existen genes en distintos organismos que guardan cierto grado de similitud (es decir, comparten total o parcialmente la secuencia) y que codifican para proteínas similares. Por ejemplo, tomemos estos fragmentos de secuencias de Citocromo C (una proteína involucrada en la respiración), pertenecientes a organismos distintos:

CitC_humano: `'AGDVEKGKKIFIMKCSQCHTVEKGGKHKTGPNLHGLFGRKTGQA'`

CitC_gorila: `'AGDVEKGKKIFIMKCSQCHTVEKGGKHKTGPNLHGLFGRKTGQA'`

CitC_pollo: `'AGDIEKGKKIFVQKCSQCHTVEKGGKHKTGPNLHGLFGRKTGQA'`

a) ¿Podés construir un programa sencillo que compare las secuencias y diga si son o no iguales?

b) Supongamos que esas secuencias están alineadas[6], de forma tal que el primer aminoácido ocupa la posición número 1 en todas las secuencias, el segundo aminoácido ocupa la posición 2 en todas, etc. ¿Cuál sería la posición menos conservada?

---

6) Dada la siguiente lista de secuencia ¿podrías construir un programa que detecte si se trata de una secuencia de ADN, ARN o proteína e imprima en pantalla de qué tipo de molécula se trata?

Sec7: `'AGDVEKGKKIFIMK'`

Sec8: `'ATTGGACGTAATGC'`

Sec9: `'AGGUACCGAUCCA'`

---

7) ¿Podrías mejorar el programa del punto anterior haciendo que sea capaz de tomar secuencias que ingresa el usuario? Recordá que no siempre el usuario utiliza mayúsculas, pero tu programa debe ser capaz de procesar tanto minúsculas como mayúsculas.

---

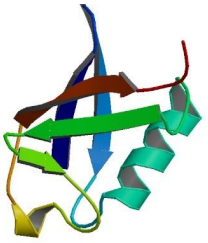
[6] Un alineamiento de secuencias es una forma de representar y comparar dos o más secuencias de ADN, ARN o de proteínas, para resaltar sus semejanzas y diferencias. Consiste en disponer las secuencias en líneas consecutivas, insertando espacios en cada una de ellas para que las zonas equivalentes queden alineadas verticalmente. Así, se asume que los aminoácidos o nucleótidos que quedan en la misma columna son equivalentes entre sí. Si dos secuencias en un alineamiento comparten un ancestro común, las diferencias en una posición dada pueden interpretarse como mutaciones puntuales (sustituciones), y los huecos en alguna de las secuencias, como fragmentos que se ganaron o perdieron durante su evolución.

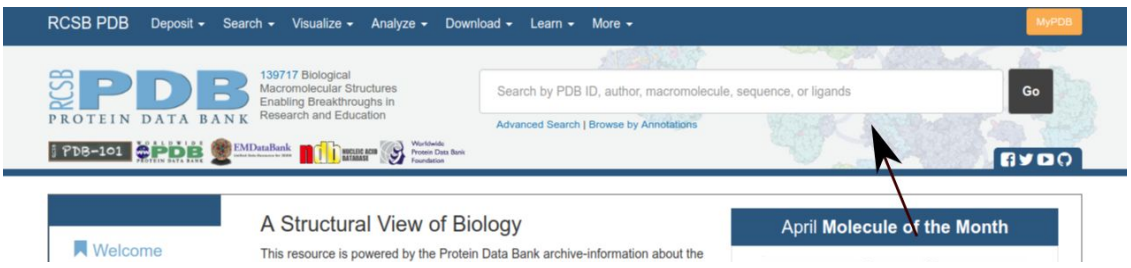
Uso y desusos de las Bases de Datos Biológicas

Hemos usado durante los talleres mucha información biológica, como secuencias de ADN, de ARN y de proteínas. ¿De dónde viene esa información? En términos generales los científicos de todo el mundo desarrollan diversos conocimientos relacionados con los seres vivos. Estos conocimientos se obtienen a base de observaciones y experimentación. Los datos y conclusiones obtenidos son compartidos entre científicos de forma organizada, ya sea por medio de publicaciones en revistas super-archi-nerds o, por ejemplo, a través de Bases de datos disponibles en internet. Una base de datos (BD) es una colección estructurada de datos; en particular, una base de datos biológica es una colección de información relacionada con seres vivos. Estos datos provienen de experimentos científicos, literatura publicada, análisis computacional, etc. La información contenida en bases de datos biológicas puede incluir, por ejemplo: funciones, estructura y localización de proteínas o genes, efectos clínicos de mutaciones, así como similitudes de secuencias o distancias evolutivas, etc. Entre las bases de datos más utilizadas por científicos de todo el mundo, bioinformáticos o no, se encuentran GenBank (colección de todas las secuencias biológicas estudiadas) y PDB (que guarda la información estructural disponible acerca de ácidos nucleicos y proteínas).

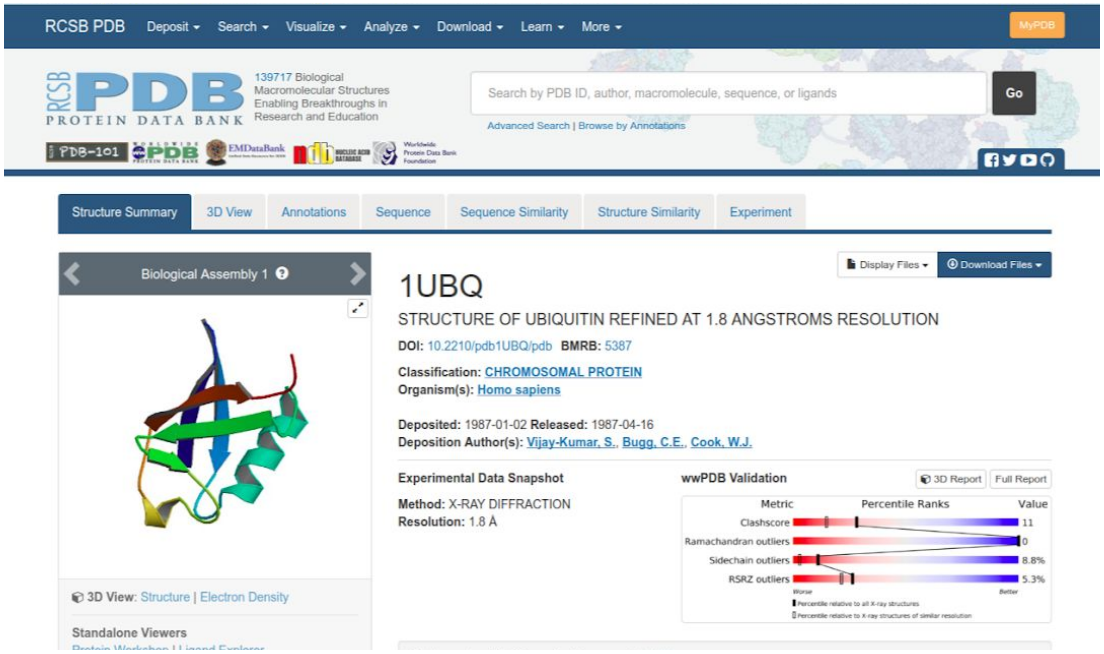
8) La ubiquitina es una proteína pequeña que ha sido encontrada en casi todas las células eucariotas (de allí viene su nombre: *ubiquo* significa omnipresente). Esta proteína es la encargada de la marcación química de las proteínas que ya no son necesarias, para que otras proteínas las reconozcan y las destruyan. (Para reflexionar: ¿Por qué una célula querría destruir sus propias proteínas?). Te invitamos a descubrir un poco más acerca de ella.

a) Desde tu celular/computadora ingresá al sitio: https://www.rcsb.org/. Esta página web corresponde a una de las bases de datos más utilizadas en la bioinformática: PDB. En esta base de datos se encuentran almacenadas todas las estructuras de macromoléculas biológicas obtenidas hasta el momento, como la presentada aquí a la derecha. Las estructuras se almacenan en forma de archivos que contienen las coordenadas en el espacio, en los ejes X, Z e Y, de todos los átomos de una molécula dada. Estas coordenadas pueden ser interpretadas por algunos programas gráficos para mostrar de forma tridimensional como se vería, por ejemplo, una proteína en una célula o en solución. En el recuadro marcado por la flecha en la imagen siguiente, ingresá el siguiente código: **1UBQ.**

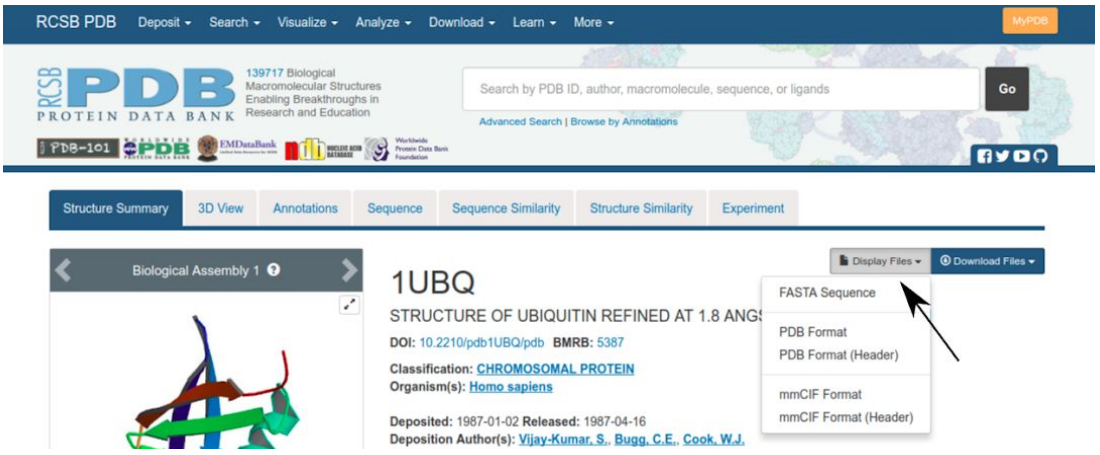




b) Se va a cargar una página con todos los datos de la ubiquitina. ¡Explorala, a ver de qué se trata!



c) Hacé click en el recuadro *'Display files'*. Se va a desplegar un pequeño menú en el que vas a seleccionar *'FASTA sequence'*.



d) Luego de hacer click ahí, se va a abrir una ventana emergente que contiene la información indicada en la siguiente imagen. Esas letras son la secuencia en formato FASTA (un formato de archivo que se usa para datos biológicos). La primera línea empieza con el símbolo mayor (>) y contiene información de la proteína, como por ejemplo su nombre. La segunda línea contiene la secuencia propiamente dicha.

```
>1UBQ:A|PDBID|CHAIN|SEQUENCE
MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLIFAGKQLEDGRTLSDYNIQKESTLHLVLRLRGG
```

Declará una variable, con el nombre que quieras, y almacená el contenido de la secuencia de ubiquitina en dicha variable. Ésta será una variable del tipo *string*.

e) ¿Podrías construir un programa que calcule el porcentaje de Leucina (L) en dicha proteína? ¿Y lo mismo, pero de Treonina (T)?

f) Ahora mejoremos el programa para que sea más "amigable con el usuario", haciéndole las siguientes mejoras:
   - Hacé que el programa le pregunte al usuario qué secuencia desea analizar.
   - Hacé que el programa consulte al usuario sobre qué aminoácido quiere calcular el porcentaje de abundancia en la proteína.
   - Hacé que el programa arroje luego la longitud de la secuencia y el porcentaje del aminoácido deseado en dicha proteína.

---

9) ¿Dado el siguiente alineamiento de secuencias, podrías generar un programa que calcule las distancias entre las secuencias (número de cambios entre dos secuencias dadas)?

a) Dadas las distancias calculadas ¿podrías agrupar las secuencias según su similitud? Entendiendo la distancia entre secuencias como el número de caracteres distintos.

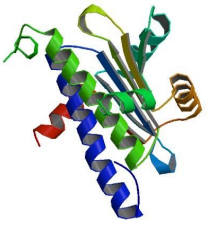
	1	2	3	4	5	6	7
Drosophila	t	t	a	t	t	a	a
fugu	a	a	t	t	t	a	a
mouse	a	a	a	a	a	t	a
human	a	a	a	a	a	a	t

---

10) La actividad biológica de una proteína depende en gran medida de la disposición espacial de su cadena polipeptídica. Las proteínas no son un ovillo, sino que adoptan una estructura dada en el espacio. Se definen cuatro niveles distintos, conocidos como estructura primaria, secundaria, terciaria, y cuaternaria, y, cada uno de ellos se constituye a partir del anterior. La estructura secundaria que una proteína adopta se debe a la formación de puentes de hidrógeno entre los átomos que forman el enlace peptídico. Los tipos básicos de la estructura secundaria son:

Amino acid	Preference α-helix	β-strand	Reverse turn
Glu	**1.59**	0.52	1.01
Ala	**1.41**	0.72	0.82
Leu	**1.34**	1.22	0.57
Met	**1.30**	1.14	0.52
Gln	**1.27**	0.98	0.84
Lys	**1.23**	0.69	1.07
Arg	**1.21**	0.84	0.90
His	**1.05**	0.80	0.81
Val	0.90	**1.87**	0.41
Ile	1.09	**1.67**	0.47
Tyr	0.74	**1.45**	0.76
Cys	0.66	**1.40**	0.54
Trp	1.02	**1.35**	0.65
Phe	1.16	**1.33**	0.59
Thr	0.76	**1.17**	0.90
Gly	0.43	0.58	**1.77**
Asn	0.76	0.48	**1.34**
Pro	0.34	0.31	**1.32**
Ser	0.57	0.96	**1.22**
Asp	0.99	0.39	**1.24**

- α-hélice: plegamiento en espiral de la cadena polipeptídica sobre sí misma.



- Lámina plegada (beta hoja plegada): el plegamiento no origina una estructura helicoidal sino una lámina plegada en zig-zag.

- Bucles: sin una forma definida.

Una misma cadena polipeptídica puede adquirir distintas estructuras secundarias, en distintos segmentos de la misma. El hecho de que una proteína adquiera una u otra estructura depende de la composición de aminoácidos que la conforman (estructura primaria), es decir que conociendo la secuencia de una proteína y la preferencia de cada uno de los 20 aminoácidos para formar parte de una u otra estructura podríamos predecir qué disposición en el espacio adoptará una dada proteína. Dada la siguiente cadena peptídica, y teniendo en cuenta la tabla de frecuencias de aparición de los distintos aminoácidos en una dada estructura secundaria:

¿Podrías construir un programa que muestre en pantalla la estructura secundaria que adoptará cada residuo, especificandola como H (si es una hélice), B (si es una hoja beta plegada) y L (si es un bucle o loop)?

Sería deseable que el programa imprimiera también la secuencia de la cadena polipeptídica en cuestión.

11) Una secuencia consenso es una secuencia formada a partir de nuestra colección de secuencias alineadas, tomando el símbolo más común en cada posición. Por supuesto, puede haber más de un símbolo más común, lo que lleva a múltiples posibles cadenas de consenso. Dadas las siguientes secuencias de proteínas, tenien

---

Nos veremos otra vez...

Hasta acá llegamos con esta guía ¡Esperamos tu participación en el próximo Concurso de Bioinformática para Escuelas Secundarias!

¡Gracias totales!

Ante cualquier duda podés contactarme por correo electrónico Ana Julia Velez Rueda (anavelezrueda@gmail.com), coordinadora del proyecto, o con cualquiera de los docentes de nuestros talleres. Encontranos también a través de la página de nuestro grupo, el SBG o Grupo de Bioinformática Estructural de la Universidad Nacional de Quilmes: http://ufq.unq.edu.ar/sbg/education/index.html.

Colaboraron con este material: Julia Marchetti, Nicolás Palopoli, Marcia Hasenahuer, Guillermo Benitez, Nahuel Escobedo, Leandro Matías Sommese, Diego Zea, Elin Teppa y Juliana Glavina.