Bioinformática 2025
TP Nº1: Entornos y Primeros Programas

TP0) Entornos de Programación

Un entorno de programación es un software/app que nos permite programar y correr los programas en un dado lenguaje (Python, c++, java, etc)
En este curso utilizaremos principalmente python como lenguaje de programación y bash (la consola de python) para correr los programas

1) Para empezar..
Un entorno fácil para empezar (de cero totalmente) en python es utilizar un entorno on-line simple que nos permita empezar a practicar algoritmos/programas/sintaxis
uno de ellos es REPLIT:

REPLIT https://repl.it/languages/python3f

Es un intérprete de python (y otros lenguajes) on-line
muy útil para practicar, probar cosas rápidas
El **registro** es opcional pero recomendado.

Otra opción muy utilizada actualmente es google colab: https://colab.google/ la cual permite integrar scripts de python con google drive. El trabajo en esta plataforma muchas veces es útil por ser colaborativo y el trabajo en grupos, pero no es recomendable trabajar al mismo tiempo.

2) Instalando python en LINUX y uso de jupyter notebook
Para usar python en nuestra PC una de las maneras que se suele utilizar es instalar python y ya sea correrlo desde la consola/línea de comando o utilizando un entorno local como jupyter.
Para ello debemos primero: **Instalar Python.**
Para Instalar Python y sus librerías se usa “PIP”

i) Lo primero entonces es Instalar Python y PIP:

Desde consola: sudo apt-get install python3-pip

Ejercicio 1:

Abrir un archivo “inicio.py” con VI(M) y poner :
Print (“Este es mi primer programa de Python”)
Cerrar y correr con: python3 inicio.py

=> LISTO HAS CREADO TU PRIMER PROGRAMA EN PYTHON

Ejercicio 2: Una vez instalado PIP se pueden instalar librerias (x ejemplo pandas)

python3 -m pip install pandas

=> haga una ronda de consulta con docentes/compañeros para ver qué librerías instalar

ii) Instalemos ahora el entorno jupyter notebook con PIP

python3 -m pip install --upgrade pip
python3 -m pip install jupyter
Se corre con: `~/.local/bin/jupyter-notebook (correrlo en el directorio de uso)`

Ejercicio 3: Corre en la Jupyter Notebook tu primer programa de Python

Si no ha tenido éxito con lo explicado en los ejercicios anteriores, le proporcionamos un colab donde se puede correr python y continuar con la guía, recomendamos luego volver a intentar con su computadora lo anterior.

Le pedimos que hagan una copia y lo vayan completando:

iii) Entornos avanzados

En algún momento de la cursada, o cuando ya la hayas terminado y quieras dedicarte a programar regularmente en python (u otros lenguajes), quieras ir un poco más allá de la Jupyter, para eso debes instalar un IDE (Integrated development environment, lo que?).

Estos programas para programar permiten varias cosas, incluyendo librerías/paquetes pre-instalado, ayuda en la sintaxis, herramientas de debug y control de versiones. Estos temas exclusivos de programación (no de bioinformática) no los vamos a ver de manera específica pero sí mencionar utilizar.

Entre los entornos e IDEs más famosos tenemos (y vas a tener que googlear un poco para elegir que usar o probarlos) Anaconda/Conda, Spyder, Vscode, PyCharm y otros. Cual utilizar es una cuestión de gustos.

IV) Ecosistema de python

El "ecosistema" de python se basa en la utilización de librerías especializadas que proveen funciones específicas.
Lo que se debe hacer es INSTALARLAS (ver PIP)
Una vez instalada hay que importarla: Import libreriaX
Las funciones se usan como: libreriaX.funcion
Uno puede cargar solo "alguna" función, se usa: from libreriaX import función
El import suele traer TODO salvo algunas librerías muy grandes en donde el import trae las funciones más comunes, y cosas específicas hay que usar import-from.

TP01) Empezando a diseñar algoritmos y programar en Python

Variables: Python es un lenguaje de programación completamente orientado a objetos. No se necesita declarar variables antes de usarlas. Cada variable en Python es un objeto y así cada objeto soporta las siguientes instrucciones:

help(object) - Muestra información de como usar objetos.

Vamos a comenzar recorriendo Números (hay de todo tipo...) y cadenas(strings) de caracteres (que son el core de los dos tipos de variables en cualquier lenguaje de programación)
Ojo: Las cadenas están definidas con comillas sencillas o compuestas.

Ejercicio 4: Utilice la consola para definir y analizar los siguientes objetos

Entero=1
Decimal=2.0
micadena="Hola"
Interrogarlos con "help"...

Nota sobre las variables: si bien al principio parece simpático y simple utilizar como nombres de variables, x, y etc.. en códigos complejos eso resulta un kaos. Una buena regla de programador es usar nombres de variables lógicos y tratar uno mismo de seguir cierta lógica o reglas de construcción de nombres.

El comando print: uno de los comandos más utilizados en python es print(), este comando le dice al programa que nos muestre lo que sea que está en la(s) variable(s) entre () en pantalla y lo podemos usar para ver que estamos guardando en ellas.

Pruebe el comando print con las variables definidas anteriormente u otras de su interés

Comentarios El carácter #:
Una buena práctica de programación es agregar al código comentarios que explican qué es lo que hace un dado bloque. En python el carácter # antes de una línea (o parte de ella hacia la derecha) le dice al intérprete que lo que sigue es un comentario y no parte del código.
A modo de ejemplo

```
x=3
print(x) # esta línea de código imprime el contenido de la variable x en pantalla
```

Pruebe poner #antes de cada una de las lineas de codigo, ¿Que pasa?

Operadores sencillos: En python (como en casi cualquier lenguaje de programación) las operaciones matemáticas (suma, resta, multiplicación, exponencial et.) y las lógicas (Mayor / Menor / Igual etc) pueden ser ejecutados de manera directa (no hay que invocar las funciones) y funcionan prácticamente sobre todo los tipos de variables. Sobre números su resultado parece bastante obvio (no?), sobre otras variables el resultado puede ser más útil o interesante.

Pruebe operaciones aritméticas entre números (de distinto tipo), entre cadenas, y mezclados?

Listas: son colecciones de elementos ordenados. Los elementos de la lista pueden ser recorridos iterativamente o accediendo a ellos vía un índice basado en cero (y esta es su utilizada).

La listas se crean explìcitamente:

```
Tipo mi_lista=[1,2,3,4,5]
```

También se la puede crear vacía []
Para verla podemos usar print(mi_lista)

Las listas permiten un montón de operaciones y funciones (`append, clear, index, count, sort, pop, insert, remove, len, max, min, etc.`)
En general se usan: lista.comando(variable)
Ejemplo: mi_lista.index(x)

Las listas se pueden recorrer usando un “loop” (existen distintos tipos de loops y ya los iremos viendo), un primer ejemplo (loop for)

```
for x in mi_lista:
        print x
```

Nota sobre Indentación: En python para ordenar la escritura de código es mandatorio cuando se utiliza un comando que luego ejecuta una serie de comandos dentro de un bloque (así sea uno solo, como en este caso) los comandos del bloque se escriben corridos un “tab” hacia la derecha. Esto se llama **indentación** y es uno de los errores de sintaxis más comunes.

Ejercicio 5: Cree una lista de números y/o elementos(caracteres), pruebe diferentes comandos para manipular la lista.

Pruebe el “for loop” para recorrer la lista. Reflexione sobre el rol de “x”
¿Qué valor toma x?

Nota sobre listas: una de las claves de las listas es que los elementos pueden ser variables en sí mismas, y por lo tanto pueden contener dentro cualquier cosa (incluso cosas distintas). Esto suena un poco “loco” y lo iremos viendo con el avance del curso, si está ansioso pruebe por ejemplo crear listas de listas, o transformar un “string” en una lista de caracteres.

Variables booleanas:
La estructura básica de cualquier algoritmo, y por ende programa, es la toma de decisiones (condicionales) y la iteración (bucles-loops). La toma de decisiones se basa en variables booleanas y lógicas (True, False, And, Or, not etc).
Estos operadores, dependen de una variable (de tipo booleana) o de una condición.

Primer ejemplo: definamos un variable booleana y hagamos un print

```
b=True o b=False
```

Segundo ejemplo: Definimos una variable y luego evaluamos condiciones

```
x = 2
print(x == 2)
print(x == 3)
print(x < 3)
```

Pruebe los comandos en su jupyter... (¿le va cazando la onda?)

Nota: para asignar valores a una variable se utiliza el operador de igualdad "=", mientras que para comparar las variables entre ellas se hace uso de dos signos de igualdad "==".
La no igualdad se realiza con el operador "!="

Toma de decisiones (Condicionales):
El uso de True - False se usa para la toma de decisiones con los comandos Si (if) entonces (then).

Ejemplo:

```
if x==2:
    print("equis es igual a 2")
```

A esto se pueden agregar operadores “booleanos” "y " (and) y "o " (or) que permite construir complejas expresiones booleanas, por ejemplo:

```
name = "John"
age = 23
if name == "John" and age == 23:
    print "tu nombre es John, y tú tienes 23 años."
```

Además podemos usar “sino(else)” o “un segundo condicional (elif) como ser:

```
if x == 2:
    print "x igual dos!"
else:
    print "x no es igual a dos."
```

Ejercicio 5: practique comandos if / else / elif con operaciones booleanas

Bucles – loops (For y While):
Hay dos tipos de loops (bucles) en Python, for y while.
Los loop For iteran sobre una secuencia (una lista predefinida o un rango)

Ejemplo:

```
        primos = [2,3,5,7]
        for x in primos:
                print(x)
```

```
for x in range(1,5):
        print(x)
```

Los loop While se repiten mientras se cumpla una condición “lógica” sea cumplida.

```
suma=0
while suma < 5:
        print(suma)
        suma = suma +1
```

Existen dos comandos básicos para controlar un loop "break" y "continue"
break es usado para salir de un loop o un loop while.
continue es usado para saltar el paso/bloque actual.

Ambos comandos son muy útiles pero cuesta cierto trabajo comprender bien su funcionamiento, practique utilizarlos un poco con loops simples para entender cómo funcionan.

Pruebe/piense la diferencia entre los siguientes códigos

```
for x in range(1,5):
    print('empezando bloque',x)
    if x == 2:
        Elija => break o continue
    print('termine bloque',x)
```

Input-Output

Una de las cosas más importantes cuando uno hace un programa código es poder enviarle/cargarle información al mismo (ya sea desde la línea de comando o desde un archivo) y luego poder obtener información (ya vimos que en línea de comando se usa el print) o en un archivo. En python entonces para ingresar (input) y sacar (output) de un programa hay típicamente dos maneras. Directa, interactiva (o por línea comando); o a través de archivos.
En el primer caso la salida ya la vimos y es el clásico “print” el equivalente de entrada es Input.

Ejemplo:

```
print('Ingrese su nombre:')
nombre = input()
print('Hola, ' + nombre)
```

Lectura / Escritura de archivos

Para leer y escribir archivos se utilizan los comandos open-read-write
Lo primero que se debe hacer es “abrir” un archivo en memoria:

file = open("nombre del file", ModoDeApertura)
Hay 4 modos de apertura:
"r" - Read - Default value. Opens a file for reading, error if the file does not exist
"a" - Append - Opens a file for appending, creates the file if it does not exist
"w" - Write - Opens a file for writing, creates the file if it does not exist
"x" - Create - Creates the specified file, returns an error if the file exists

Para leerlo una vez abierto se usa “read”.
read() lo lee entero, si queremos leer línea x línea usamos readlines() , luego debemos cerrarlo con close.
Ejemplo:

```
f = open("file_de _prueba", "r")
print(f.readline())
f.close()
```

Cuando abrimos un archivo con “a” append “w” esto nos permite agregar líneas y/o sobreescribir el archivo:

Pruebe por ejemplo los siguientes códigos

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
```

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()
```

Finalmente para abrir un archivo nuevo se usa “x” en este caso si el archivo existe devuelve un error. Al igual que el read y el write se puede hacer por líneas y con formatos (esto deberá explorarlo por su cuenta)

Funciones
Para que el código no sea un caos, además de utilizar los bloques de código por indentación, uno puede crear lo que se llaman funciones (antes subrutinas) que son pequeños bloques de código (algoritmos) que hacen una función específica y toman un “input” y entregan un “output”
Para entender cómo es su sintaxis mire y pruebe el siguiente código:

```
# definición de la función
def mi_funcion(var1,var2):
    output=var1+var2
    return output
```

```
#uso de la función
x=mi_funcion(a,b)
print(x)
```

Ejercicio 6: Pruebe jugar cambiando quienes son a y b (diferentes enteros, flotantes, o caracteres), pruebe cambiar la función, por ejemplo reemplazando la suma por otra cosa.

Primeros Ejercicios programados:

A continuación se presentan 4 ejercicios que permitirán realizar los primeros programas e integrar lo aprendido en este primer módulo o para los que ya saben programar practicar un poco.

Ejercicio 1:

1) Obtén todos los números primos del 1 al 20 usando loops y condicionales, e imprímelos.
2) Extiende el código para determinar los números primos hasta un límite definido por el usuario.

Trate primero de entender en términos computacionales que significa que un número sea primo.

3) Piense entonces que deberia hacer un codigo para identificar un número primo. Escriba entonces un grafico que permita entender que es lo que hará su codigo (pseudocodigo).

4) Existe una función/operación módulo (o resto). Piense si le puede ser util para el código.

Recuerde que para recorrer elementos en orden se usan listas.

5) Una vez tenga alguna primera versión del programa funcionando:
   5a) Analice la performance en tiempo (cuanto tarda su programa?). Use graficos de tiempo vs número de predicciones.
   5b) Evalue como mejorar la performance. Haga un pseudocodigo, y si encuentra como, implementelo. Haga un grafico de tiempo para comparar con 5a.

Ejercicio 2:

Crea un programa que para generar posibles passwords use Nombre, Fecha de nacimiento y DNI.
a) Para ello utilice las herramientas para que el código le pida que ingrese estos datos y le de una clave. Investigue el uso de diccionarios, listas y permutaciones.

Requisitos: 8 caracteres, al menos 2 letras, 2 números y sin fragmentos del nombre, fecha o DNI.

Ejercicio 3:

Selecciona o crea un archivo de texto y cuenta cuántas veces aparece una palabra dada (ingresada por el usuario).

Cuente la frecuencia de cada palabra y luego represente el conteo en una nube de palabras (puede explorar la librería WordCloud o alguna que considere útil).

Pruebe el código con algún archivo de texto, ¿para qué podría usar esa representación? ¿le parece necesario excluir alguna palabra para realizar el gráfico? ¿Considera que está usted sesgado en la utilización de alguna muletilla/palabra al escribir?

Ejercicio 4:

Ya que vamos a intentar programar nuestros módulos bioinformáticos para ganar en versatilidad y para emparejar con el otro track de la materia, ¿podría programar una búsqueda en una base de datos primaria ?

¿Qué características tiene una base de datos primaria (BD1) ? ¿Qué querría buscar en ellas ?

Usando algo de información de https://biopython.org/docs/dev/Tutorial/chapter_entrez.html, podría construir un programa que baje un registro de una base de datos usando palabras clave ? autor, año, revista, base de datos.

¿Puede buscar palabras clave dentro de esos registros ?

¿Qué campos tiene ese registro ?