[Unidad 1: Introducción a la programación en Python](#_mu0gu553375i)

[1. Introducción](#_3ya0zjs7j3tr)

[2. Breve historia del lenguaje de programación Python](#_h309b1ryxya5)

[3. Versiones e intérpretes disponibles](#_hnp8w96i2glg)

[4. Sintaxis básica del lenguaje de programación Python](#_7chkod1xc12p)

[4.1. Intérprete frente a script](#_q9qmz8qx2zk5)

[4.2. Sobre tabuladores y espacios en blanco](#_db7bufm9jn5u)

[4.3. Guía de estilo](#_c5rsokuyyn8c)

[4.4. IPython, Jupyter, Notebook y Google Colab](#_mbrvlllgcbuz)

[5. Ejercicios y preguntas teóricas](#_awx5mlb3f68w)

[5.1. Materiales](#_2nf38nqu6icn)

[5.2. Entrega e instrucciones importantes](#_hzzpt49g0gqw)

[6. Bibliografía complementaria](#_hbjpiacv5ojr)

# Unidad 1: Introducción a la programación en Python

## 1. Introducción

Encontramos en Wikipedia la siguiente definición para **programación**:

*"La* **programación informática** *o* **programación algorítmica***, acortada como* **programación***, es el proceso de diseñar, codificar,* [*depurar*](https://es.wikipedia.org/wiki/Depuraci%C3%B3n_de_programas) *y mantener el* [*código fuente*](https://es.wikipedia.org/wiki/C%C3%B3digo_fuente) *de* [*programas computacionales*](https://es.wikipedia.org/wiki/Programas_inform%C3%A1ticos)*. El código fuente está escrito en un* [*lenguaje de programación*](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n)*. El propósito de la programación es crear programas que exhiban un comportamiento deseado."*

En el contexto de las ciencias de la vida y la salud, los ordenadores son máquinas que nos permiten tratar, procesar y analizar ingentes cantidades de datos. Este tratamiento de la información debe realizarse mediante algún tipo de conjunto de reglas: el algoritmo. Podemos entender los algoritmos como plantillas o *recetas* que se especifican a la máquina mediante algún lenguaje de programación y que nos permiten tratar la información de forma inherente a nuestras necesidades. Siguiendo con la metáfora de la receta, imaginad que queremos preparar un plato de comida y que tenemos todos los ingredientes para ello. Nuestra receta culinaria especificará en cada paso lo que el incipiente cocinero debe realizar. Por ejemplo, para una tortilla:

1. Romper los huevos y batirlos.

2. Poner al fuego una sartén.

3. Cuando la temperatura esté lo suficientemente alta, echar los huevos batidos.

4. Esperar a que cuaje lo suficiente

5. Dar la vuelta.

6. Repetir el paso 4 por esta cara.

7. Emplatar nuestra flamante tortilla.

Los problemas que resolveremos mediante un ordenador normalmente tendrán un contexto más numérico, pero la idea de receta será la misma: un conjunto de instrucciones que se ejecutan desde el principio hasta el final y que realizan alguna acción. Así pues, el algoritmo o *receta* para calcular el cuadrado de un número cualquiera sería la siguiente:

1. Representar por pantalla el mensaje "Bienvenido a la calculadora de cuadrados:".

2. Pedir al usuario que introduzca mediante el teclado un número.

3. Guardar este número.

4. Utilizar el número guardado para multiplicarlo por sí mismo.

5. Representar por pantalla el resultado del cálculo del paso 4.

6. Salir del programa.

Especificar a un ordenador las instrucciones que deben ejecutarse en un programa es una abstracción que nos permiten los lenguajes de programación. En nuestro caso, utilizaremos Python para escribir y ejecutar nuestros propios programas y así solucionar problemas de carácter bioinformático.

## 2. Breve historia del lenguaje de programación Python

El lenguaje nació de la mano de Guido van Rossum mientras trabajaba en el Centro para las Matemáticas y la Informática (CWI, Centrum Wiskunde & Informatica) en los Países Bajos, en el año 1989 durante las vacaciones de Navidad, aunque no fue hasta febrero de 1991 cuando fue publicada su primera versión en [USENET](https://es.wikipedia.org/wiki/Usenet).

[Guido van Rossum](https://www.python.org/~guido/) es, pese al largo recorrido del lenguaje desde entonces, una figura muy relevante en el grupo de usuarios y desarrolladores de Python y es conocido como **BDFL** (del inglés *Benevolent Dictator for Life*, Benevolente Dictador Vitalicio, aunque ha abandonado el puesto en julio de 2018) y sus sugerencias en las discusiones en las que interviene son tomadas muy en cuenta respecto a la hoja de ruta del futuro del lenguaje. En la actualidad, la **Python Software Foundation** ([PSF](https://www.python.org/psf/)) es la organización encargada de velar por el futuro y la diseminación del lenguaje. Como nota divertida, cabe destacar que la relación entre el nombre del lenguaje y la serie de televisión [Monty Python's Flying Circus](https://www.imdb.com/title/tt0063929/) no es una mera coincidencia.

Dos fechas importantes han marcado la historia del lenguaje: el 16 de octubre de 2000, momento en el que la versión 2.0 fue liberada, y el 3 de diciembre de 2008, cuando lo fue la versión 3.0 del lenguaje.

El lenguaje de programación Python hace especial énfasis en su capacidad para ser leído por otros programadores. Esta aparente sencillez a la hora de leer código en Python le ha permitido gozar de bastante popularidad en círculos menos enfocados en el desarrollo puro de software, como es el campo de la bioinformática, en el que confluyen diferentes expertos en distintas disciplinas (física, química, biología, matemáticas, estadística, etc.). Existen documentos de mejora del lenguaje conocidos por sus siglas **PEP-Número** (del inglés *Python Enhancement Proposals*) y es, precisamente, en uno de estos documentos, el [PEP-20](https://legacy.python.org/dev/peps/pep-0020/), en el que se presenta la filosofía del lenguaje y está marcada tendencia por la simplicidad, legibilidad y belleza del código en Python:

The Zen of Python  
    Beautiful is better than ugly.  
    Explicit is better than implicit.  
    Simple is better than complex.  
    Complex is better than complicated.  
    Flat is better than nested.  
    Sparse is better than dense.  
    Readability counts.  
    Special cases aren't special enough to break the rules.  
    Although practicality beats purity.  
    Errors should never pass silently.  
    Unless explicitly silenced.  
    In the face of ambiguity, refuse the temptation to guess.  
    There should be one~~and preferably only one~~obvious way to do it.  
    Although that way may not be obvious at first unless you're Dutch.  
    Now is better than never.  
    Although never is often better than \*right\* now.  
    If the implementation is hard to explain, it's a bad idea.  
    If the implementation is easy to explain, it may be a good idea.  
    Namespaces are one honking great idea ~~let's do more of those!~~

Muchos de estos principios en este momento podrán parecer extraños al alumno, pero a medida que se profundiza en el uso del lenguaje muchos de ellos resultan habituales a la hora de escribir y leer código en Python.

Por último, es interesante presentar al alumno el término *pythonic* pues aparece en muchas búsquedas en relación a la sintaxis de Python. Los lenguajes de programación hacen uso habitualmente de estructuras que sirven para expresar tareas o partes de código en un lenguaje común que no está considerado directamente en la sintaxis del lenguaje. Este concepto (en inglés *idiom*) está muy extendido en la comunidad de Python y muchas veces se traduce en mensajes en foros del estilo “*which is the most pythonic way to do..?*” (*¿cuál es la forma más pythonic de hacer..?*). Se trata de un concepto avanzado, pero que en Python resulta especialmente relevante puesto que algunos de estos *idioms* han sido implementados en el lenguaje de forma más eficiente y pueden hacer que un código u otro se ejecute de forma más rápida si es más pythonic. Este concepto será ampliado más adelante.

## 3. Versiones e intérpretes disponibles

En la actualidad coexisten dos versiones o ramas de desarrollo: **las familias 2.7 y 3.9**, siendo sus últimas versiones estables en el momento de redacción de este documento la 2.7.18 y la 3.9.6, respectivamente.

Python se trata de un lenguaje de programación muy vivo y en continua evolución, al que se le van añadiendo y corrigiendo funcionalidades muy a menudo. La rama de desarrollo 2.7.x es una versión muy madura y estable, pero fue con la aparición de la versión 3.0 cuando hubo un salto importante en la sintaxis del lenguaje. Por ejemplo, la expresión propia *print* en la rama 2.7.x pasó a tener forma de función *print()* en la rama 3.x. Estos cambios no se limitan a correcciones estéticas, sino que se han introducido para mejorar el rendimiento del lenguaje, la organización del código y de las librerías a nivel interno, etc. En resumen, la rama 3.x es el futuro de Python mientras que la rama 2.7.x representaría el pasado del lenguaje.

La versión 2.7.x ha tenido un amplio recorrido, es madura y existe abundante documentación en foros, y muchas librerías a nivel científico y bioinformático aún no han sido traducidas a la versión 3.x de Python. Mientras que las más relevantes ya lo han sido, algunas no y quizá nunca lo serán. En cualquier caso, el código de la versión 3.x que presentaremos durante el curso puede ser fácilmente traducido a su versión 2.7 sin demasiado esfuerzo.

En cuanto a los intérpretes, el más conocido es la implementación en el lenguaje de programación [C](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n)) llamado [**CPython**](https://es.wikipedia.org/wiki/CPython), que es el intérprete del lenguaje Python desarrollado por la comunidad del proyecto de código libre y soportado y dirigido por la [Python Software Foundation](https://www.python.org/psf/). Aunque CPython se trate de la implementación estándar y que a día de hoy soporta más plataformas (más de 40 en el momento de redacción de este documento), no es ni mucho menos la única. Otras famosas implementaciones son [Jython](http://www.jython.org/), escrito en [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)) y que corre en la máquina virtual de Java, [PyPy](https://es.wikipedia.org/wiki/PyPy), un intérprete también escrito en el lenguaje C, pero que pone especial énfasis en el rendimiento y en tratar la concurrencia y el paralelismo de una forma diferente, [IronPython](https://es.wikipedia.org/wiki/IronPython), una implementación del lenguaje para la plataforma .NET, etc.

## 4. Sintaxis básica del lenguaje de programación Python

### 4.1. Intérprete frente a *script*

Existen dos formas básicas de ejecutar código en Python (el carácter $ indica el comando ejecutado en la terminal):

1. Ejecutar el intérprete de Python desde una terminal:

| $ python3 Python 3.8.8 (default, Mar 18 2021, 06:01:57)  [Clang 11.0.3 (clang-1103.0.32.62)] on darwin  Type "help", "copyright", "credits" or "license" for more information.  >>> 2 \* 3  6  >>> |
| --- |

1. Escribir código Python en un archivo (típicamente con extensión “.py”, y que es conocido habitualmente con el nombre de *script*) y ejecutarlo utilizando el comando:

| $ python3 nombre\_de\_mi\_script.py |
| --- |

La primera forma es útil para probar comandos del lenguaje Python de forma rápida o simplemente para utilizar el intérprete como calculadora (probad a escribir la expresión 2+2\*3). La segunda, por el contrario, nos permitirá ejecutar una y otra vez el código contenido en nuestro fichero de *script*sin tener que escribir en el intérprete el mismo código una y otra vez.

### 4.2. Sobre tabuladores y espacios en blanco

En Python, los bloques de código se indican mediante espacios en blanco. Más adelante hablaremos de los diferentes tipos de bloques de código, pero a modo introductorio utilizaremos el ejemplo siguiente:

| 1 def suma(a, b): 2 return a + b 3 4 print(suma(2, 3)) |
| --- |

En las líneas 1 y 2, definimos mediante la palabra clave def una función que recibe como argumentos dos variables de nombre a y b y que en la línea 2 devuelve la suma de esas dos variables.

En la línea 4, llamamos a esa función *suma* con los valores 2 y 3, y mediante la función print()escribimos el resultado que nos aparece por terminal.

Como se puede observar en la línea 2, la palabra clave return se encuentra a cuatro espacios de distancia del comienzo de la línea anterior (línea 1). Esto es así puesto que estamos definiendo un bloque de código (en otros lenguajes como C o Java, se utilizan para la definición de bloques de código los caracteres especiales “{“ y “}”).

Existen dos posibilidades para definir la indentación de código (*indentación* es un anglicismo y en español deberíamos utilizar el término sangrado, aunque indentación está ampliamente reconocido por la comunidad informática): mediante espacios o mediante tabulaciones. Podremos utilizar de forma indistinta una u otra forma, pero siempre teniendo en cuenta que **jamás deberán mezclarse** en un mismo fichero de código. Esto es así porque las tabulaciones no poseen un carácter que las distinga y no suelen presentarse de forma idónea en muchos editores de texto. En muchas ocasiones al presionar la tecla “Tab” en una aplicación, este carácter se convierte automáticamente en un número fijo de espacios en blanco. Por lo tanto, **será recomendable configurar el editor que escojamos para escribir código en Python de tal forma que el tabulador se traduzca en 4 espacios en blanco**.

En el documento [PEP-8](https://www.python.org/dev/peps/pep-0008/), que es la propia guía de estilo que siguen la mayoría de programadores Python, se detalla este aspecto ([tabs-or-spaces](https://www.python.org/dev/peps/pep-0008/#tabs-or-spaces).)).

### 4.3. Guía de estilo

En el apartado anterior se ha tratado la guía de estilo oficial del lenguaje Python, que está disponible en línea como [PEP-8](https://www.python.org/dev/peps/pep-0008/). A modo de resumen, estos son los aspectos básicos a seguir en este curso:

1. Las funciones y los nombres de palabras se escribirán en minúscula y los nombres largos que contengan diversas palabras se unirán mediante el carácter “\_”.
2. Los nombres de clases tendrán su letra inicial en mayúscula para diferenciarlas de variables, funciones y objetos.
3. Las constantes irán en mayúsculas y separadas por el carácter “\_” si contienen diversas palabras.
4. Como ya se comentó en el apartado anterior, no se utilizarán tabuladores, sino que siempre se usarán cuatro espacios en blanco.

Es muy importante seguir la guía de estilo en esta unidad y en las siguientes, pudiendo verse afectada de forma negativa la nota obtenida si no se sigue la guía de estilo.

### 4.4. IPython, Jupyter, Notebook y Google Colab

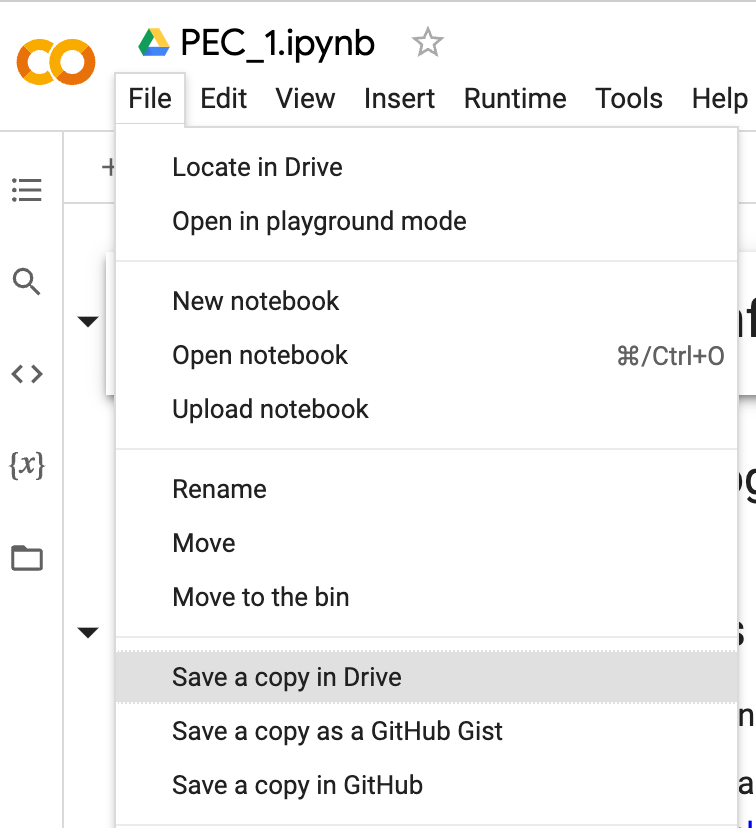
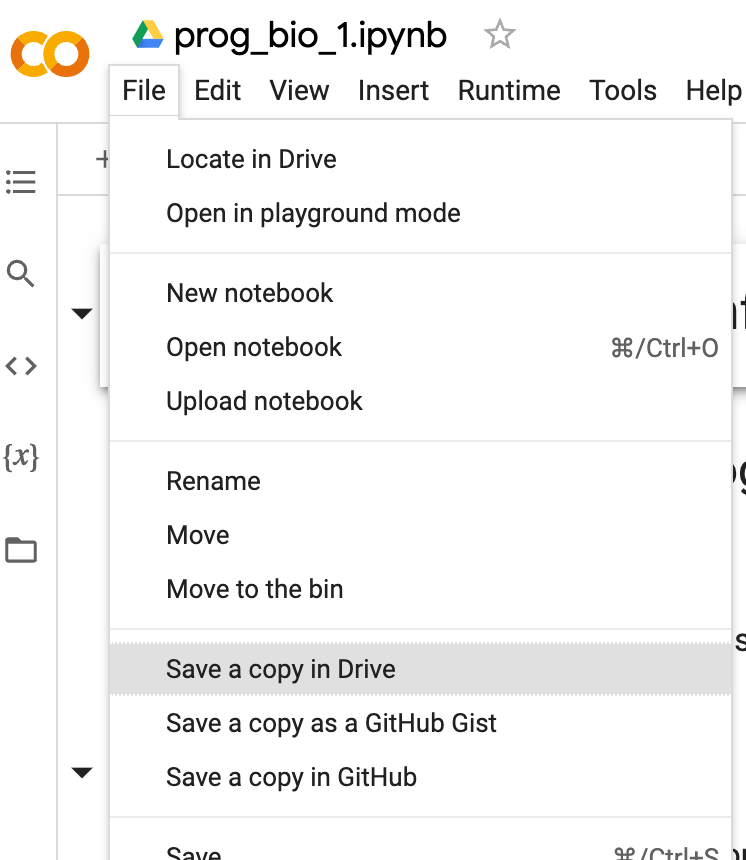
[IPython](http://ipython.org/) fue originalmente desarrollado como un conjunto de herramientas para facilitar el desarrollo y el procesamiento de datos en Python a usuarios del mundo científico. La “i” del nombre proviene de *interactivo*, puesto que dispone de herramientas que interactúan con el usuario, como son el autocompletado de código (del que dispone ahora el intérprete de Python), una extensa ayuda, etc. [Notebook](http://ipython.org/notebook.html), a su vez, es otra herramienta muy útil para la exploración de código, para representar resultados de forma gráfica, o para el prototipado rápido, etc., todo esto mediante la interacción vía navegador web (Notebook arranca de forma local un servidor web que se encarga de ejecutar el código Python que escribamos vía el navegador). Las últimas versiones de IPython Notebook ahora se conocen como [Jupyter](https://jupyter.org/).

Desde hace un tiempo, Google también ofrece el servicio de *notebooks* en la nube mediante su plataforma [Google Colab](https://colab.research.google.com). En este curso vamos a hacer uso de Google Colab.

## 5. Ejercicios y preguntas teóricas

La parte evaluable de esta unidad consiste en la entrega de un archivo Jupyter-Notebook de nombre PEC\_1.ipynb que contendrá los diferentes ejercicios y las preguntas teóricas que hay que contestar. También debéis entregar el fichero prog\_bio\_1.ipynb este contiene ejemplos para realizar PEC y no será evaluado. Para poder cargar, modificar y ejecutar los ficheros haremos doble clic sobre el archivo desde el navegador en la carpeta de Google Drive correspondiente y este nos redirigirá automáticamente a la plataforma Google Colab. Si el doble click no funciona, hacemos click con el botón derecho y despues abrir con Google Collab.

Si no veis la opción de Google Collab, clicad en *connect more apps* > *Google collaboratory* y después ya podréis abrir los ficheros con doble click o con boton derecho y despues abrir con Google Collab.

Una vez abierto el fichero en Google Colab debéis seleccionar “Save a copy in Drive” en el menú “File”: 

### 

### 

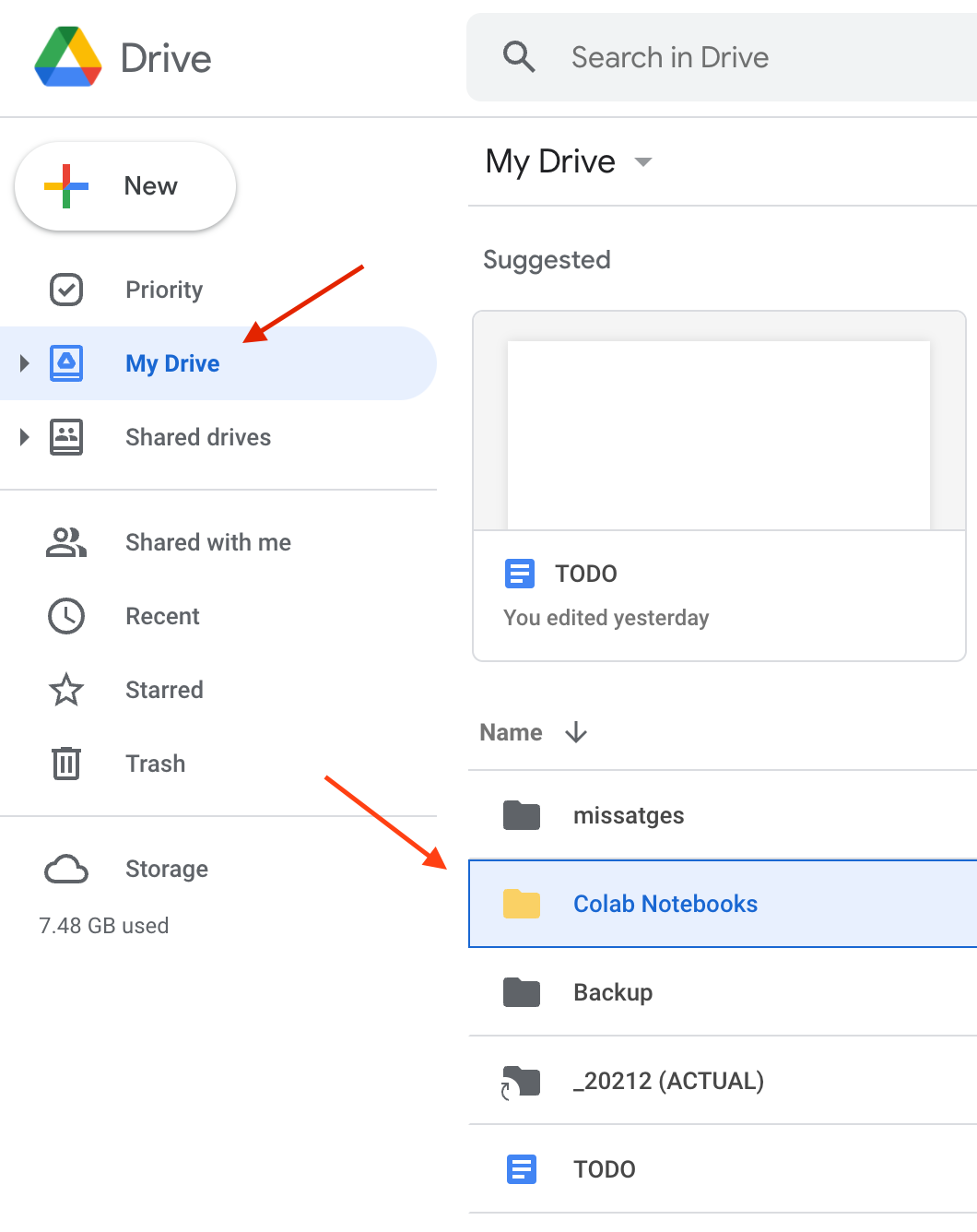
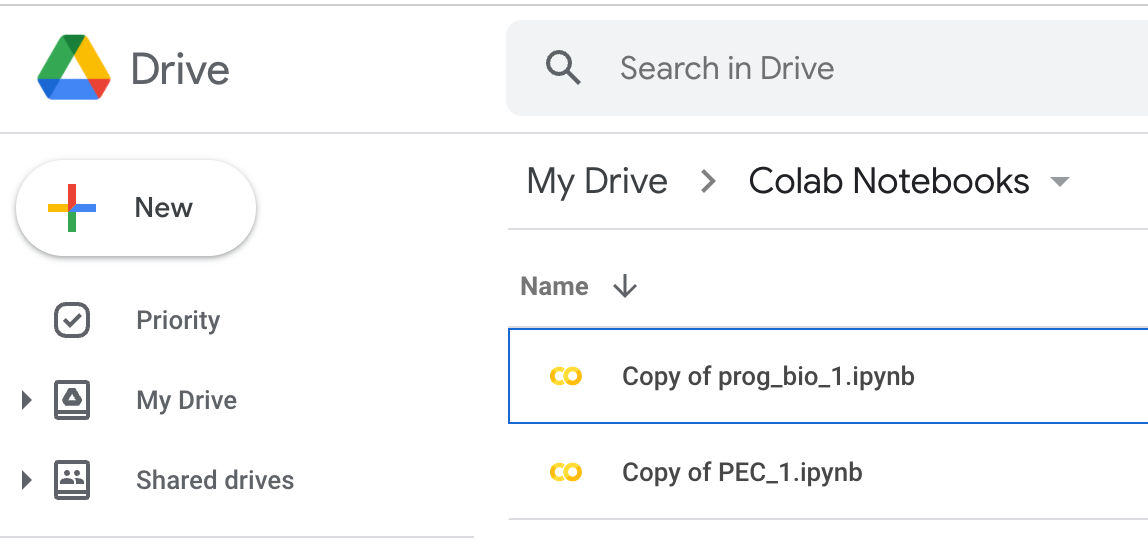
### 

### 

### 

### 

A continuación ambos fichero aparecerán en la carpeta “Colab Notebooks” de “My Drive” y podréis trabajar con ellos:



### 

### 

### 

### 

### 

### 

### 

### 

En el fichero de ejercicios podréis encontrar tanto ejercicios prácticos como preguntas teóricas. Cada uno de ellos tendrá una etiqueta que indica los recursos necesarios para resolverlos. Hay tres posibles etiquetas:

* **MU** **Materiales unidad**: las herramientas necesarias para realizar la actividad se pueden encontrar en los materiales de la asignatura.
* **EG** **Consulta externa guiada**: la actividad puede requerir utilizar herramientas que no se encuentran en los materiales de la asignatura, pero el enunciado contiene indicaciones de como encontrar la información adicional necesaria.
* **CI** **Consulta externa independiente**: la actividad puede requerir utilizar herramientas que no se encuentran en los materiales de la asignatura, y el enunciado no incluye las indicaciones de como encontrar la información adicional. El estudiante deberá buscar esta información independientemente.)

### Estos ejercicios están dentro de la sección *ejercicios evaluables*. Además de estos ejercicios, hay otros de gran utilidad que no tienen etiquetas y están dentro de la sección llamada *ejercicios ejemplo*. Esta sección contiene ejercicios similares a los evaluables. Estos ejercicios son de entrenamiento y podréis comentar su resolución en el foro de la asignatura. El hecho de comentar estos ejercicios en el foro, junto con la participación en el mismo, computan el 5% de la nota que se indica en el plan docente como participación activa en el aula. Estos ejercicios ejemplo, a pesar de no ser evaluables, si no están resueltos al entregar la PEC supondrá una penalización a la nota final de los evaluables.

### 5.1. Materiales

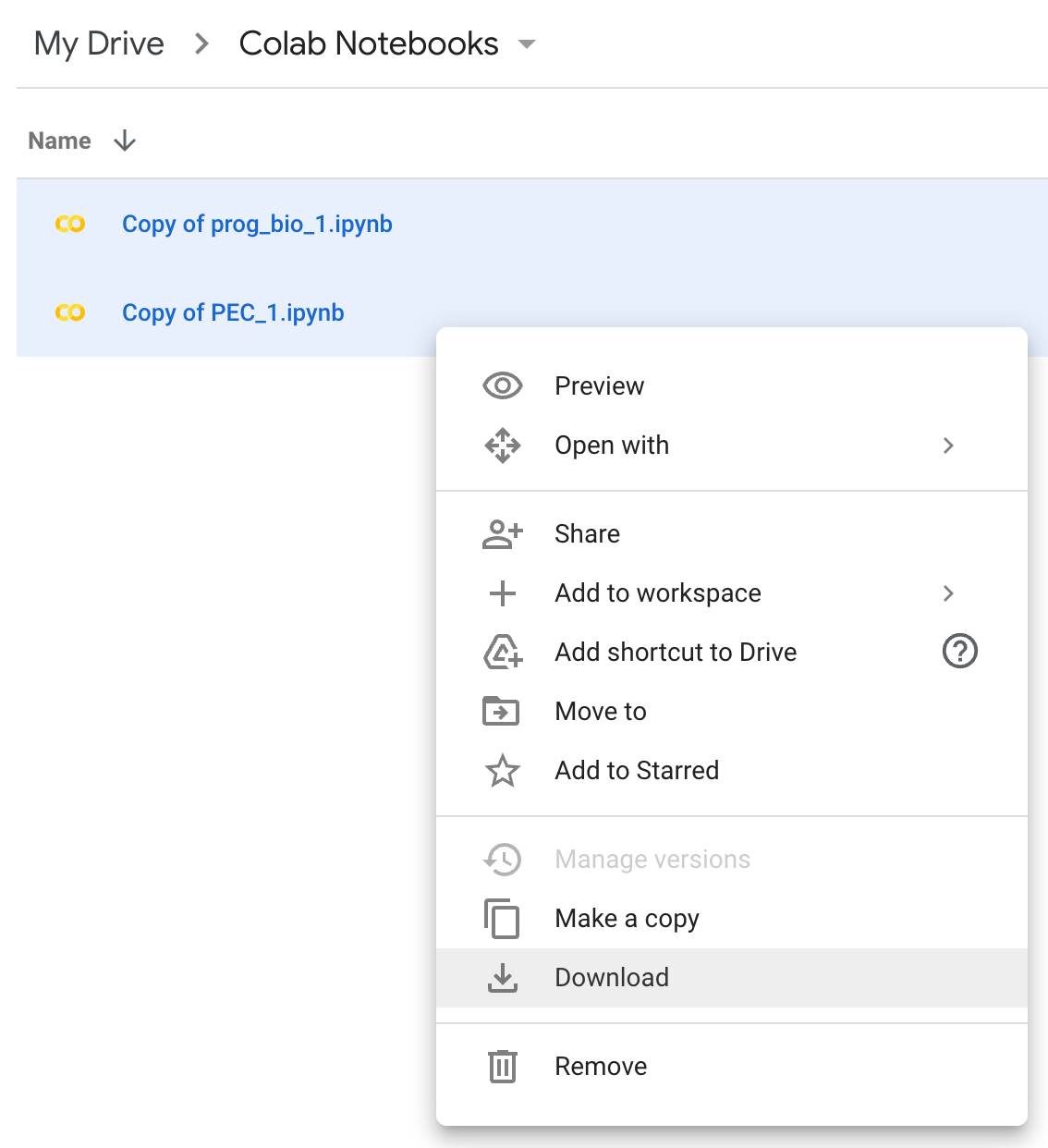
* Fichero notebook de la teoría: prog\_bio\_1.ipynb
* Fichero notebook de ejercicios: PEC\_1.ipynb
* Vídeo-tutorial sobre tipos de variable: variables.mp4
* Vídeo-tutorial sobre strings y listas: string\_listas.mp4

**Nota sobre los vídeo-tutoriales: Estos vídeos fueron realizados para la versión 2.7 de Python y utilizando IPython Notebook en vez de Google Colab como hacemos ahora, pero el código es 100% compatible salvo en el uso del comando print, que ahora se corresponde con la función print(). Los encontraréis en la carpeta de esta unidad.**

### 5.2. Entrega e instrucciones importantes

Recordad que vuestras soluciones deben ser originales. **Esperamos NO detectar copia directa entre estudiantes**. En caso de copia, será notificado al director de estudios del máster.

Para hacer la entrega, seleccionad ambos ficheros, clic en el botón secundario de vuestro ratón y seleccionad la opción “Download” (Descargar):



Una vez descargado el archivo de vuestra solución comprimido en formato zip, podréis subirlo al apartado de entrega correspondiente, en este caso *LLiurament PAC1 | Entrega PEC1*, para que podamos evaluarlo. Si hacéis cambios en el notebook de la solución, podéis volver a repetir los pasos y subir tantas versiones como sean necesarias. **Utilizaremos la última versión a la hora de corregir vuestra PEC**.

## 6. Bibliografía complementaria

1. [Bassi, Sebastian Python for Bioinformatics](https://discovery.biblioteca.uoc.edu/discovery/fulldisplay?docid=alma991000619449706712&context=L&vid=34CSUC_UOC:VU1&lang=ca&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&isFrbr=true&tab=Everything&query=any,contains,Python%20for%20bioinformatics&offset=0)
2. [Youens-Clark, Ken Mastering Python for Bioinformatics](https://discovery.biblioteca.uoc.edu/discovery/fulldisplay?docid=alma991000734484706712&context=L&vid=34CSUC_UOC:VU1&lang=ca&search_scope=MyInst_and_CI&adaptor=Local%20Search%20Engine&tab=Everything&query=any,contains,mastering%20python%20for%20bioinformatics&offset=0)
3. [Google Colaboratory FAQ](https://research.google.com/colaboratory/faq.html)