Laboratorio 0: Python, Configuración y Tutorial de Autoevaluador

Introducción

Los laboratorios de esta clase utilizan Python 3.

El Laboratorio 0 cubrirá lo siguiente:

- Instrucciones sobre cómo configurar Python.
- Ejemplos de flujo de trabajo.
- Un mini-tutorial de Python.
- Evaluación del proyecto: Cada entrega de proyecto incluye un autoevaluador (autograder) que se puede ejecutar localmente para corregir errores y realizar un proceso de retroalimentación de cierta manera autónomo.

Archivos para Editar y Entregar

Durante la asignación, deberá completar partes de los archivos **addition.py**, **buyLotsOfFruit.py** y **shopSmart.py** que se encuentran en el archivo **tutorial.zip**.

Una vez que haya completado la asignación, deberá subir capturas de pantalla con el resultado del *script* de autoevaluación y el código fuente con las soluciones con el nombre de archivo tutorialnombreestudiante.zip, por ejemplo: ***tutorialArlesRodriguez.zip***

Nota Adicional

Este laboratorio ha sido elaborado originalmente por la Universidad de Berkeley y puede utilizarse con fines académicos. Los autores insisten en no compartir las soluciones por internet.

Como se cuenta con acompañamiento y a nivel local se realiza una adaptación de los laboratorios basada en el material del curso, se recomienda realizar los laboratorios a conciencia con miras a aprender a implementar los algoritmos. Cuentan con el apoyo del profesor y la idea, como mencionan los autores, es que el laboratorio sea una experiencia amena y enriquecedora.

Instalación de Python

Necesitas una distribución de **Python (3.6 o superior)** y, además, **Pip** o **Conda**.

Para verificar que cumples con los requisitos, abre la terminal y ejecuta **python -V**. Deberías ver una versión que sea suficientemente alta. Luego, ejecuta **pip -V** y **conda -V** para asegurarte de que al menos uno de ellos funciona e imprime la versión de la herramienta.

Recomendaciones de Instalación

Si necesitas instalar alguna herramienta, por simplicidad, se recomienda **Python 3.9 y Pip**. Si ya tienes Python, solo necesitas instalar Pip.

Si optas por usar **Conda** a través de **Anaconda** o **Miniconda**, estas ya incluyen Python y Pip, por lo que solo tendrías que instalar un único paquete. Esta es la instalación que encontrarás en las salas de cómputo: **Guía de instalación de Anaconda**.

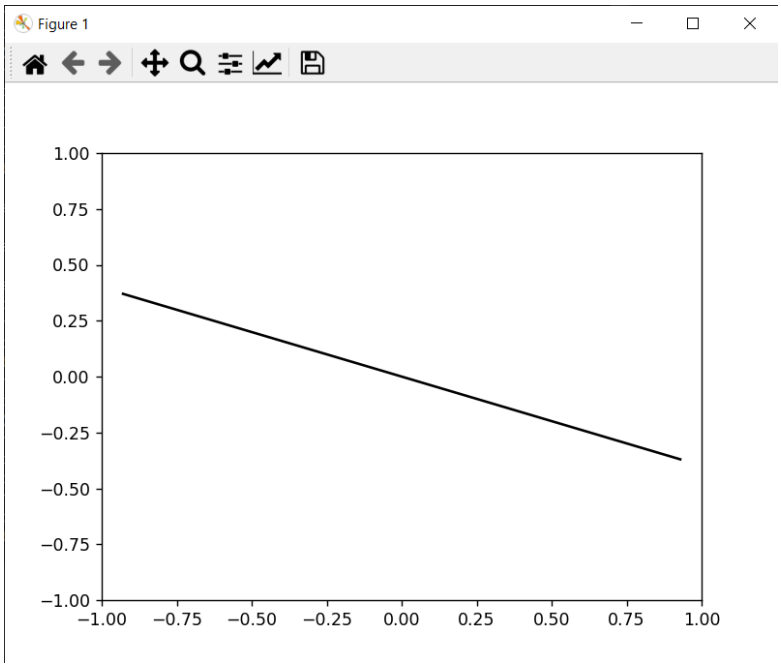
Instalación de Dependencias

El proyecto de aprendizaje automático tiene dependencias adicionales. Es importante instalarlas ahora para que, si hay algún problema con la instalación de Python, no tengamos que volver atrás o rehacer la instalación más tarde.

En las instalaciones de **Conda**, las dependencias ya deberían estar incluidas. Puedes verificar que todo funciona correctamente ejecutando el siguiente comando en la terminal:

```
python autograder.py --check-dependencies
```

Si la instalación es correcta, debería aparecer una ventana emergente similar a la imagen donde se ve un segmento de línea girando en círculo.



Solución de Problemas

En caso de no observar la ventana emergente

Las librerías necesarias y los comandos correspondientes para instalarlas son:

- **numpy**: Proporciona soporte para arreglos multidimensionales grandes y rápidos.
- **matplotlib**: Es una librería para la creación de gráficos 2D.

Ejecuta los siguientes comandos en la terminal para instalarlas:

```
pip install numpy
pip install matplotlib
```

Después de la instalación, utiliza el comando `python autograder.py --check-dependencies` para confirmar que todo funciona correctamente.

Verificación de instalaciones

En algunas instalaciones, es posible que los comandos python3 y pip3 se refieran a las versiones que necesitas. Además, puede haber múltiples instalaciones de Python, lo que podría complicar dónde se instalan los paquetes.

Usa los siguientes comandos para verificar tu entorno:

- **`python -V`**: Muestra la versión de Python.
- **`pip -V`**: Muestra la versión de Pip y la ruta de instalación de Python a la que está asociado.
- **`which python`**: Muestra la ubicación de la instalación de Python.

Error de importación de tkinter

Si encuentras un error de importación de tkinter, es probable que tu instalación de Python no sea la típica y provenga de Homebrew. La solución es desinstalar esa versión y luego instalarla de nuevo a través de Homebrew, asegurándote de incluir el soporte para tkinter. Alternativamente, puedes usar el instalador gráfico recomendado para una instalación más sencilla.

Ejemplos de Flujo de Trabajo y Configuración

No se espera que uses un editor de código en particular, pero aquí tienes algunas sugerencias sobre flujos de trabajo convenientes. Puedes revisarlos rápidamente por medio minuto y elegir el que te parezca mejor:

- **GUI e IDE, con atajos de VS Code**: Se recomienda encarecidamente leer la sección **"Uso de un IDE"** si planeas usar uno, para que aprendas a utilizar sus funciones más convenientes.
- **En la terminal, usando comandos de Unix y Emacs**: Este método funciona bien incluso en Windows. Es útil si necesitas editar código en cualquier máquina sin necesidad de una configuración previa, o si trabajas con conexiones remotas.

Evaluación mediante autograde

Para familiarizarte con el autoevaluador, te pediremos que programes, pruebes y entregues tu código después de resolver tres preguntas.

Puedes descargar todos los archivos asociados al tutorial de autoevaluación como un archivo ZIP: **`tutorial.zip`**

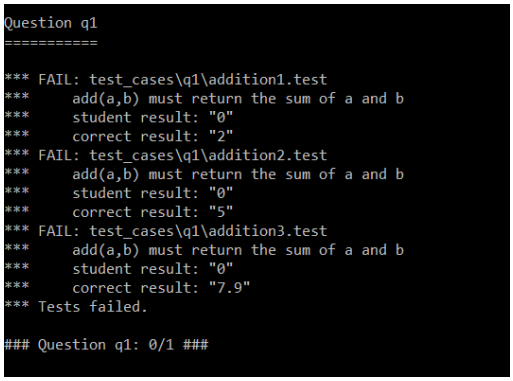
El archivo contiene una serie de archivos que editarás o ejecutarás:

- `addition.py`: archivo fuente para la pregunta 1.
- `buyLotsOfFruit.py`: archivo fuente para la pregunta 2.
- `shop.py`: archivo fuente para la pregunta 3.
- `shopSmart.py`: archivo fuente para la pregunta 3.
- `autograder.py`: script de autoevaluación (ver más abajo).

También hay otros archivos que puedes ***ignorar*** por ahora:

- `test_cases`: directorio que contiene los casos de prueba para cada pregunta.
- `grading.py`: código del autoevaluador.
- `testClasses.py`: código del autoevaluador.
- `tutorialTestClasses.py`: clases de prueba para este proyecto en particular.
- `projectParams.py`: parámetros del proyecto.

El comando **`python autograder.py`** califica tu solución a los tres problemas. Si lo ejecutas antes de editar cualquier archivo, obtendrás una o dos páginas de resultados.



Para cada una de las tres preguntas, verás los resultados de las pruebas, la calificación de la pregunta y un resumen final. Como aún no has resuelto las preguntas, todas las pruebas fallarán. A medida que resuelvas cada pregunta, es posible que algunas pruebas pasen mientras otras fallan. Cuando todas las pruebas de una pregunta pasen, obtendrás la puntuación completa.

Al observar los resultados de la pregunta 1, verás que ha fallado tres pruebas con el mensaje de error: "`add(a, b) must return the sum of a and b`". Esto indica que la respuesta que tu código da es siempre 0, pero la respuesta correcta es diferente. Solucionaremos esto en la siguiente sección.

Pregunta 1: Suma

Abre el archivo **`addition.py`** y revisa la definición de la función `add`:

```
def add(a, b):
    "Return the sum of a and b"
    "*** YOUR CODE HERE ***"
    return 0
```

Las pruebas llaman a esta función con a y b con diferentes valores, pero el código siempre devuelve cero. Modifica esta definición para que quede de la siguiente manera:

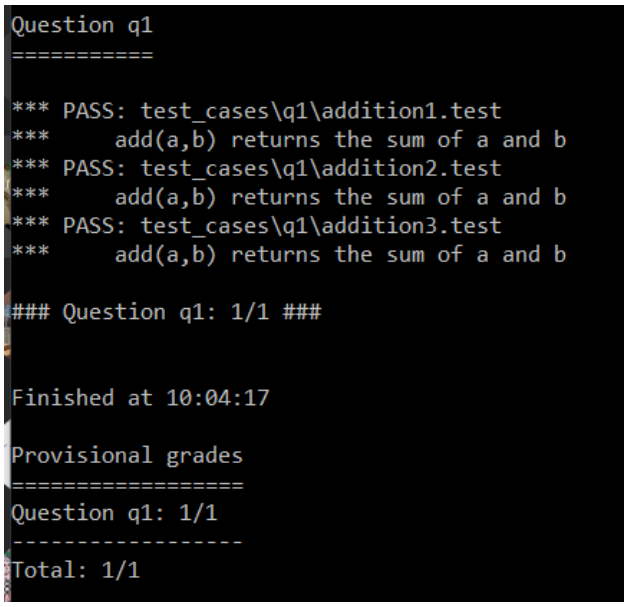
```
def add(a, b):
    "Return the sum of a and b"
    print("Passed a = %s and b = %s, returning a + b = %s" % (a, b, a + b))
    return a + b
```

Ahora, vuelve a ejecutar el autoevaluador para la pregunta 1 (omitiendo los resultados de las preguntas 2 y 3) con el siguiente comando:

```
python autograder.py -q q1
```

Ahora deberías pasar todas las pruebas y obtener la puntuación máxima para la pregunta 1. Fíjate en las nuevas líneas ***"Passed a=..."*** que aparecen antes de **`"* PASS: ..."`**. Estas son generadas por la instrucción `print` dentro de la función `add`. Puedes usar este tipo de instrucciones `print` para mostrar información útil que te ayude a depurar tu código.

La salida del código anterior debería verse como la siguiente:



Pregunta 2: Función buyLotsOfFruit

Implementa la función **`buyLotsOfFruit(orderList)`** en el archivo **`buyLotsOfFruit.py`**. Esta función debe recibir una lista de tuplas con el formato **`(fruit, numPounds)`** y devolver el costo total de la lista de frutas.

Si una fruta en la lista no se encuentra en el diccionario **`fruitPrices`**, la función debe imprimir un mensaje de error y devolver **`None`**. Por favor, **no modifiques la variable `fruitPrices`.**

Ejecuta el comando `python autograder.py -q q2` hasta que la pregunta 2 pase todas las pruebas y obtengas la puntuación completa. Cada prueba verificará que `buyLotsOfFruit(orderList)` devuelva la respuesta correcta para diferentes entradas.

Por ejemplo, el archivo `test_cases/q2/food_price1.test` verifica si:

El costo de **`[('apples', 2.0), ('pears', 3.0), ('limes', 4.0)]`** es 12.25.

Pregunta 3: Función shopSmart

Completa la función `shopSmart(orderList, fruitShops)` en el archivo **`shopSmart.py`**. Esta función recibe una **`orderList`** (similar a la que se le pasa a `FruitShop.getPriceOfOrder`) y una lista de **`FruitShop`**. La función debe devolver la instancia de `FruitShop` donde el costo total del pedido es el más bajo.

Por favor, no cambies el nombre del archivo ni de las variables. Ten en cuenta que te proporcionaremos la implementación de **`shop.py`** como un archivo de apoyo que puede utilizarse para resolver el problema más fácilmente.

Ejecuta el comando **`python autograder.py`** hasta que la pregunta 3 pase todas las pruebas y obtengas la puntuación completa. Cada prueba confirmará que `shopSmart(orderList, fruitShops)` devuelve la respuesta correcta para diferentes entradas posibles.

Por ejemplo, con las siguientes variables definidas:

```
orders1 = [('apples', 1.0), ('oranges', 3.0)]
orders2 = [('apples', 3.0)]
dir1 = {'apples': 2.0, 'oranges': 1.0}
shop1 =  shop.FruitShop('shop1',dir1)
dir2 = {'apples': 1.0, 'oranges': 5.0}
shop2 = shop.FruitShop('shop2', dir2)
shops = [shop1, shop2]
```

- **`test_cases/q3/select_shop1.test`** verifica si: `shopSmart.shopSmart(orders1, shops) == shop1`

- **`test_cases/q3/select_shop2.test`** verifica si: `shopSmart.shopSmart(orders2, shops) == shop2`

Envío:

Una vez que hayas completado el laboratorio, deberá subir capturas de pantalla con el resultado del *script* de autoevaluación y el código fuente con las soluciones con el nombre de archivo tutorialnombreestudiante.zip, por ejemplo: ***tutorialArlesRodriguez.zip***

Referencias:

Project 0. (s. f.). CS 188 Fall – University of California, Berkeley 2025. https://inst.eecs.berkeley.edu/~cs188/fa25/projects/proj0/