Tecnología Digital 1: Introducción a la Programación

# Trabajo Práctico 1

Escuela de Negocios, UTDT - Primer semestre de 2022

*Antes de comenzar, se recomienda resolver el Ejercicio 9 de la Guía 4, para entender los sistemas de numeración y cómo cambiar de base.*

Decimos que un número $n \in \mathbb{N}$ (es decir, un entero mayor que 0) es *cafetero* si y solo si su representación hexadecimal contiene exactamente una ocurrencia de los símbolos C, A, F y E, en ese orden, y no contiene ocurrencias de los símbolos B y D. Sí puede tener cualquier cantidad de ocurrencias de los dígitos 0 al 9 en cualquier posición, incluso intercalados entre los símbolos C, A, F y E.

Por ejemplo, $51966$ es el menor número cafetero; su representación hexadecimal es `CAFE`. Los números $827902$ (`CA1FE`) y $29994894$ (`1C9AF8E`) también son cafeteros. Por otra parte, **no** son cafeteros los números $45036$ (`AFEC`) porque el orden de los símbolos es incorrecto; $831470$ (`CAFEE`) porque está repetido el símbolo E; y $831467$ (`CAFEB`) porque incluye al símbolo B.

El objetivo de este trabajo es crear un programa en Python que ofrezca las siguientes funcionalidades:

- dado un string $s$, devolver un nuevo string formado exactamente por las letras `C`, `A`, `F` o `E` que aparezcan en $s$, en el mismo orden de ocurrencia;
- dado $n \in \mathbb{N}$, determinar si $n$ es cafetero;
- dado $n \in \mathbb{N}$, obtener el $n$-ésimo número cafetero, considerando al $51966$ como el $1$-ésimo número cafetero;
- dados $n, m \in \mathbb{N}$, con $n \leq m$, obtener la lista de números cafeteros entre $n$ y $m$, inclusive en ambos casos (es decir, mayores o iguales a $n$, y menores o iguales a $m$), ordenada de menor a mayor.

Para cada una de estas funcionalidades requeridas, se pide:

a) Escribir la especificación de una función que resuelva el problema.

b) Construir un conjunto de casos de test, que se consideren suficientes para ilustrar y poner a prueba el comportamiento esperado.

c) Pensar un algoritmo y llevarlo a un programa en Python que resuelva el problema.

d) Asegurarse de que la función pasa los casos de test construidos.

e) Justificar por qué los programas escritos hacen lo esperado. En particular, demostrar informalmente que los ciclos terminan y son correctos.

Al ejecutar el programa se deberá desplegar un menú desde el cual seleccionar la funcionalidad a la que se desea acceder. Dependiendo de la opción elegida y los argumentos ingresados, el programa tendrá que mostrar la respuesta que corresponde. A continuación puede observarse parte de una ejecución del programa en la cual el usuario consulta si el número $12345$ es o no cafetero:

```
Funciones disponibles
---------------------
A. Filtrar solo CAFE [s]
B. Es cafetero [n]
C. N-ésimo cafetero [n]
D. Cafeteros entre [n,m]
X. Finalizar

Seleccione una opción: B

Ingrese n: 12345
El 12345 no es cafetero.

Presione ENTER para continuar.
```

La ejecución debe concluir únicamente si se selecciona la opción `Finalizar`; caso contrario, tiene que desplegar nuevamente el menú de opciones.

En la siguiente tabla se ilustran los mensajes que debe imprimir el programa para algunas funciones y valores ingresados:

| Función seleccionada | Argumento(s) | Mensaje por pantalla |
|---|---|---|
| `Filtrar solo CAFE [s]` | `ABcdEF12` | `String filtrado: 'AEF'.` |
| `Filtrar solo CAFE [s]` | `AÁaA` | `String filtrado: 'AA'.` |
| `Es cafetero [n]` | `12345` | `El 12345 no es cafetero.` |
| `Es cafetero [n]` | `51966` | `El 51966 es cafetero.` |
| `N-ésimo cafetero [n]` | `1` | `El 1-ésimo cafetero es el 51966.` |
| `N-ésimo cafetero [n]` | `5` | `El 5-ésimo cafetero es el 314110.` |
| `Cafeteros entre [n,m]` | `1,100` | `Cafeteros entre 1 y 1000: []` |
| `Cafeteros entre [n,m]` | `99999,200000` | `Cafeteros entre 99999 y 200000: [117502, 183038]` |

Se deben entregar los siguientes cuatro archivos:

- **`cafetero.py`**, con el código fuente de todas las funciones implementadas: `filtrar_solo_CAFE`, `es_cafetero`, `n_esimo_cafetero`, `cafeteros_entre`, y cualquier otra función auxiliar que se defina. Es obligatorio especificar el tipo de todas las variables y funciones mediante sugerencias de tipos (*type hints*). Las especificaciones de las funciones se deben incluir con el formato de *docstrings* presentado en clase.
- **`cafetero-testing.py`**, con los tests de unidad de todas las funciones definidas.
- **`tp1.py`**, con el código principal para ejecutar el programa como se indica arriba. Este código es el encargado de imprimir el menú, invocar a las funciones definidas en `cafetero.py` y mostrar los resultados por pantalla.
- **`informe.pdf`**, un documento con *(i)* la justificación de los casos de test elegidos (es decir, por qué son buenos candidatos para revelar la presencia de errores); y *(ii)* las respuestas al punto e) enunciado arriba. Incluir cualquier aclaración adicional que se considere necesaria sobre cualquier parte del trabajo. Se espera que este documento sea conciso, de no más de cinco páginas, y en formato PDF.

La carpeta adjunta `templates` contiene archivos de ejemplo para usar como referencia.

**Observaciones y aclaraciones:**

- El trabajo se debe realizar en grupos de **tres estudiantes**. La entrega consistirá en trabajo original realizado íntegra y exclusivamente por quienes integran el grupo.
- La fecha límite de entrega es el **jueves 14 de abril a las 23:55**. Los TPs entregados fuera de término serán aceptados hasta 48h pasada esta fecha, pero la demora incidirá negativamente en la nota.
- Los archivos deben subirse a la *Entrega del TP1* en la página de la materia en el campus virtual.
- Las únicas bibliotecas que se permite importar son `typing` (para *type hints*), y `unittest` (para correr los tests en `cafetero-testing.py`).
- La función `print` solamente puede invocarse en el cuerpo principal del código, en el archivo `tp1.py`. Es decir, no puede usarse en las funciones requeridas ni en las funciones auxiliares.
- Sólo pueden usarse las instrucciones y estructuras de control vistas en clase. En particular, no pueden usarse iteradores (`for`).
- Para obtener la representación hexadecimal de un número natural, se permite usar la función `hex` de Python, que devuelve un string con formato `'0x1d97c'`. Puede además usarse la función `upper` para pasar ese string a mayúsculas. Por ejemplo: `hex(121212).upper()` devuelve `'0X1D97C'`. No es necesario especificar, implementar ni demostrar las funciones `hex` y `upper` en este TP.

- Puede suponerse que el usuario siempre invocará las funciones de manera correcta. Es decir, si hay errores de tipo o valor de los argumentos provistos, no se espera comportamiento alguno del programa (podría colgarse o terminar en un error, por ejemplo).
- Con respecto al punto e) enunciado arriba, se pide demostrar terminación y correctitud solamente de las funciones que tengan algún ciclo. En el caso de que una función sin ciclos utilice a una función auxiliar que contenga un ciclo, solo deberán hacer las demostraciones sobre la función auxiliar.
- **Forma de evaluación:**
  - 50 % de la nota corresponde al *código*. Se evalúa no solo su correcto funcionamiento, sino también que se sigan las buenas prácticas de programación vistas desde la primera clase: uso adecuado de tipos y estructuras de control, claridad en los nombres de variables y funciones auxiliares, comentarios, reutilización de funciones, etc.
  - 30 % corresponde a las *especificaciones y demostraciones* de terminación y correctitud;
  - 20 % corresponde al *testing* de las funciones.