UNIVERSIDAD TORCUATO DI TELLA

Métodos Analíticos Aplicados a los Negocios II

Primer Cuatrimestre 2020

TP1

Recomendaciones generales

Esta guía contiene un Trabajo Práctico de MAAN II. El mismo deben ser resuelto íntegramente en `Python`. La resolución no es simple y se espera que resolverlo tome cierto tiempo por fuera de las clases.

Recuerden que pueden consultarlos sin ningún tipo de problema y que este trabajo y las siguientes son parte de la evaluación que tendrán de la materia. Sean responsables, no se copien. Si se detecta copia, toda la guía tendrá un 0 sin excepción. Se sugiere que empiecen a resolver el TP con tiempo de modo de tener tiempo para hacer consultas, no andar apurado a último momento.

El Trabajo Práctico debe resolverse de **forma individual**. Las consultas pueden hacerse en persona ya sea después o antes de clases o pasando por mi oficina, previa coordinación de horario. Obviamente se van a responder en el momento, con detalle y mucha paciencia. Las consultas también podrán realizarse por mail, y posiblemente se pida acordar algún horario de consulta presencial. Tener en cuenta que si la consulta es realizada a último momento puede no ser factible responderla.

Ejercitando fundamentos básicos de programación

El objetivo de este trabajo práctico es ejercitar los conceptos básicos vistos hasta el momento resolviendo ejercicios simples. Incorporar estos conceptos es *fundamental* para poder avanzar hacia problemas de mayor complejidad.

Indicaciones para la resolución

Junto con este enunciado se adjunta el archivo `tp1.py` con el esqueleto básico de cada una de las funciones pedidas, incluyendo algunos casos de prueba para cada una de ellas. Las reglas básicas para la resolución del TP son las siguientes:

- No es posible modificar la signatura de las funciones provistas, ya sea agregando o quitando parámetros.
- No es posible utilizar módulos que reemplacen la funcionalidad pedida por la función.
- Sí es posible agregar módulos para funciones particulares (ejemplo, calcular una raíz cuadrada).
- Sí es posible (y en algunos casos, **recomendable**) agregar funciones auxiliares.
- Sí es posible agregar comentarios que faciliten la lectura del código.

Enunciado

Se pide resolver los siguientes ejercicios:

1. **(0.5 puntos)** Implementar la función `minimo`, que toma dos números y retorna el mínimo de ellos. Adicionalmente, implementar la función `minimo_3`, que toma tres números (en lugar de dos) y devuelve el mínimo entre los tres. *El puntaje máximo se obtendrá si `minimo_3` está definida solamente en función de `minimo`.*

2. **(0.5 puntos)** Implementar la función `contar_pares`, que dada una lista de números enteros (`int`) retorna la cantidad de elementos pares en la misma. La lista recibida como parámetro **no** debe modificarse.

3. **(0.5 puntos)** Implementar la función `posicion_minimo_lista`, que toma como parámetro una lista y devuelve la posición del elemento más chico de la misma. La lista recibida como parámetro **no** debe modificarse. En caso de haber más de un mínimo, cualquier posición es válida.

```
>>> l = [3, 4, 1, 5]
>>> pos = posicion_minimo_lista(l)
>>> pos
2
```

4. **(0.5 puntos)** Implementar la función `media`, que toma como parámetro una lista de números y retorna el promedio de los mismos. La lista recibida como parámetro no debe modificarse. Dados $x_1, x_2, \ldots, x_n \in \mathbb{R}$, la media es

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

5. **(1 puntos)** Implementar la función `reverso`, que toma una lista `l` como parámetro y devuelve una nueva lista `l1` donde `l1[i] = l[n-1-i]`. La lista recibida como parámetro **no** debe ser modificada.

```
>>> l = ['a','a','c','b']
>>> l1 = reverso(l)
>>> l1
['b', 'c', 'a', 'a']
```

6. **(1 puntos)** Implementar la función `es_capicua`, que toma una lista como parámetro y devuelve `True` si la misma es *capicúa*, y `False` en caso contrario. La lista recibida como parámetro **no** debe ser modificada. *El puntaje máximo se obtendrá si se utiliza la función reverso.*

```
>>> l1 = [1,2,3] # No es capicua.
>>> l2 = ['a','b','b','a'] # Es capicua.
```

7. **(1 puntos)** Implementar la función `concatenar`, que toma dos listas como parámetro y devuelve una nueva lista con la concatenación de ambas. Las listas recibidas como parámetro **no** deben ser modificadas.

```
>>> l1 = [1,2,3]
>>> l2 = [4,5,6]
```

```
>>> l = concatenar(l2, l1)
>>> l
[4,5,6,1,2,3]
```

8. **(1 puntos)** Implementar la función `sumar_listas`, que toma dos listas de números como parámetro y devuelve una nueva lista con la suma *elemento a elemento* de las mismas. Las listas recibidas como parámetro no deben ser modificadas. **Asumir que ambas listas tienen la misma longitud y al menos un elemento**.

```
>>> l1 = [1,2,3]
>>> l2 = [4,5,6]
>>> l = sumar_listas(l1, l2)
>>> l
[5,7,9]
```

9. **(1 puntos)** Implementar el *procedimiento* `quitar_apariciones`, que dada una lista y un elemento borra de la lista todas las apariciones del mismo. Los cambios deben ser aplicados sobre la lista recibida como parámetero.

```
>>> l = ['a','b','b','a']
>>> quitar_apariciones(l, 'b')
>>> l
['a', 'a']
```

10. **(1 puntos)** Implementar la función `ordenar_lista`, que toma una lista como parámetro y retorna una nueva lista ordenada ascendentemente. La lista recibida como parámetro **no** debe modificarse. **No está permitido usar funciones built-in de `python` que realicen la operación.**

```
>>> l = [3, 5, 1, 2, 3, 5, 6]
>>> l1 = ordenar_lista(l)
>>> l1
[1, 2, 3, 3, 5, 5, 6]
>>> l
[3, 5, 1, 2, 3, 5, 6]
```

Algunas sugerencias (no obligatorias y dependen de la solución adoptada):

- Recordar que las listas son *mutables*, y por lo tanto si se modifica el parámetro recibido dentro de la función afectará su valor fuera de la misma. Una opción es *dentro de la función* realizar una *copia del parámetro*, y modificar la copia. Ejemplo:

```
>>> l = [1,2,3]  # Tenemos la lista l
>>> l1 = list(l) # Al usar list(), crea una copia.
>>> l1[0] = 300   # Como hicimos una copia, l no deberia
    modificarse
>>> l1
[300,2,3]
```

```
>>> l
[1 ,2 ,3]
```

- Puede ser de utilidad usar la función `posicion_minimo_lista` como parte de la solución.

11. **(2 puntos)** El *número de fibonacci* se define de la siguiente manera:

$$\begin{aligned} F(0) &= 0, \\ F(1) &= 1, \\ F(n) &= F(n-1) + F(n-2) \text{ si } n > 1. \end{aligned}$$

Implementar las siguientes funciones:

a) `fibo_recursiva`, que calcula el número de fibonacci usando recursión;

b) `fibo_iterativa`, que calcula el número de fibonacci de manera iterativa. *Sugerencia: pensar que representa la k-ésima iteración del ciclo.*

Evaluación

La resolución del trabajo consiste en completar el código para las funciones pedidas, agregando aquellas funciones auxiliares que consideren necesarias. Un 10 % del puntaje de cada ejercicio será asignado a la calidad de los comentarios incluídos en el texto, los nombres de variables y de funciones elegidos. También se considerará positivamente el agregado de *casos de test* adicionales en la función `main`.

Modalidad de entrega

Además del código con las implementaciones es posible entregar un mini informe de no más de tres carillas en donde detallen, en caso de considerarlo necesario, decisiones que tomaron para resolver uno o más ejercicios y que ayuden a entender la estrategia de resolución.

Fechas de entrega

Formato Electrónico: **viernes 17 de Abril, hasta las 23:59 hs.**, enviando el trabajo (informe + código) a las direcciónes `maan2utdt@gmail.com`. El subject del email debe comenzar con el texto [TP1 MAAN II] seguido de la lista de apellidos de los integrantes del grupo. Todos los integrantes del grupo deben estar copiados en el email.

Importante: El horario es estricto. Los correos recibidos después de la hora indicada serán considerados re-entrega.