Fundamentos de Algoritmos – Guía de Ejercicios

Master in Management + Analytics, Escuela de Negocios, UTDT

Primer semestre 2020

1. Expresiones, asignaciones y primeros programas en Python

Ejercicio 1.1. Evaluar **a mano** las siguientes expresiones y determinar cuál es su tipo. Después, revisar evaluándolas en la consola `ipython`, y averiguar su tipo con la operación `type()`.

(a) `1 + 1`

(b) `'ho' + "la"`

(c) `(1>0) or (not ('a'<'b'))`

(d) `len('hola') + 6`

(e) `(5.6 >2.0) and (len('hola')<2)`

(f) `1 + 1.0`

(g) `11 / 2`

(h) `11 // 2`

(i) `1 / 0`

(j) `123 + '123'`

(k) `1 + int('123')`

(l) `str(123) + '123'`

Comparar el resultado de evaluar las expresiones (g) y (h). ¿A qué se deben los errores de las expresiones (i) y (j)? ¿Qué ocurrió exactamente en las expresiones (k) y (l)?

Ejercicio 1.2. La operación `round()` redondea un número real al entero más cercano. Por ejemplo, `round(1.9)` devuelve 2. Evaluar `round(0.5)`, `round(1.5)`, `round(2.5)`, `round(3.5)`, etc. Averiguar a qué se debe el comportamiento observado.

Ejercicio 1.3. ¿Qué problema puede tener la comparación == entre dos valores de tipo `float`? ¿Cómo se puede solucionar? ¿Cómo podría afectar a las comparaciones `!=`, >= y <=?

Ejercicio 1.4.

(a) Escribir el siguiente programa en un archivo holamundo.py:

```
1 print("hola mundo")
```

Ejecutar el programa y observar su salida, ya sea con el botón ▶ en Spyder, o bien con el comando "`python holamundo.py`" en la consola del sistema operativo (*Terminal*, *Command Prompt*, etc.).

(b) Reemplazar el texto del programa por el siguiente y observar qué sucede al intentar ejecutar el programa.

```
1 print("hola
2 mundo")
```

(c) Reemplazar el texto del programa por el siguiente y observar la salida de su ejecución.

```
print("hola\nmundo\nchau\tmundo")
```

(d) Reemplazar el texto del programa por el siguiente y observar la salida de su ejecución.

```
# Esto es un comentario.
print("""En Python se pueden
escribir strings en varias lineas
""")
```

¿Qué se puede concluir de los símbolos """ y # en Python?

Ejercicio 1.5. En el ejercicio anterior apareció el caracter \. Se conoce a la barra invertida como *caracter de escape*, y su función dentro de una cadena de caracteres es darle un significado especial al caracter que figura inmediatamente a continuación. Por ejemplo, \n se usa como caracter de nueva línea. Escribir las siguientes instrucciones en la consola interactiva `ipython`:

```
print("\"\n")
print("""Cuidado con las "comillas".""")
print('Tengamos "Mucho cuidado" por favor.')
```

Ejercicio 1.6. ¿Qué valor se imprime en la última instrucción del siguiente programa?

```
a = 1
b = a
a = 2
print(b)
```

¿Por qué no imprime 2, si en la segunda instrucción indicamos que `b` valiera lo mismo que `a`?

Ejercicio 1.7. ¿Qué se imprime en las instrucciones 2 y 4 del siguiente programa? Entonces, ¿cuál es el tipo de la variable `a` en este programa?

```
a = 1
print(type(a))
a = 'hola'
print(type(a))
```

Ejercicio 1.8. Escribir un programa `saludador.py` que se comporte de la siguiente manera:

```
$ python saludador.py Juan
Hola Juan

$ python saludador.py Alicia
Hola Alicia
```

Observación: Como este programa recibe un argumento, es necesario ejecutarlo desde la consola del sistema operativo (*Terminal*, *Command Prompt*, etc.).

Ejercicio 1.9. Escribir un programa que tome como argumento una temperatura expresada en grados Fahrenheit, y la convierta a grados Celsius. Ejemplos:

```
$ python temperatura.py 95.1
95.10 grados Fahrenheit equivalen a 35.06 grados Celsius.

$ python temperatura.py 0
0.00 grados Fahrenheit equivalen a -17.78 grados Celsius.
```

Observación: Como este programa recibe argumentos, es necesario ejecutarlo desde la consola del sistema operativo (*Terminal*, *Command Prompt*, etc.).

2. Control de flujo

Ejercicio 2.1.

(a) Escribir un programa en Python que imprima por pantalla una tabla de conversión de millas a kilómetros (1 mi $\simeq$ 1,61 km). La tabla debe contener sólo la parte entera de la conversión, empezar en 0 km, terminar en 100 km e imprimir los valores tomando intervalos de 10 km. Es decir, por pantalla debe imprimirse:

```
  0   0
 10   16
 20   32
 ...
100   161
```

(b) Modificar el programa del punto (a) de modo tal que imprima un encabezado sobre la tabla que indique la unidad de medida de cada columna.

(c) Repetir el punto (b), pero con la conversión inversa: de kilómetros a millas.

(d) Repetir el punto (b), pero imprimiendo los valores con dos decimales en la columna correspondiente a las millas.

Ejercicio 2.2.

(a) Escribir un programa en Python que imprima los primeros n números naturales, uno por línea. El argumento n debe leerse de la línea de comandos.

(b) Repetir el punto (a), pero para los primeros n números impares.

Ejercicio 2.3. Escribir un programa en Python que dado un número $n > 0$ (ingresado por línea de comandos) imprima por pantalla un triángulo de asteriscos de altura n. No está permitido usar el operador $*$ de strings. Por ejemplo, para $n = 5$ debería imprimir:

```
*
* *
* * *
* * * *
* * * * *
```

Ejercicio 2.4. Escribir programas en Python que dado un número n calculen:

(a) $n!$

(b) $\frac{1}{n!}$

(c) $\sum_{i=1}^{n} i$

(d) $\sum_{i=1}^{n} \frac{1}{i}$

Ejercicio 2.5. Escribir un programa en Python que determine si un número dado es *perfecto*. Es decir, si es igual a la suma de sus divisores propios positivos. Por ejemplo, el 6 y el 28 son perfectos, pues $6 = 1 + 2 + 3$ y $28 = 1 + 2 + 4 + 7 + 14$. Probarlo para los primeros 10000 números naturales; deben encontrarse exactamente cuatro números perfectos.

Ejercicio 2.6. Escribir un programa en Python que, dada una palabra, determine si se trata de un palíndromo (palabra capicúa). Por ejemplo:

- entrada: `anilina` → salida: "`sí`"
- entrada: `harina` → salida: "`no`"

Sugerencia: utilizar la función `len(x)` para obtener la cantidad de letras del string `x`, y el operador `x[i]` para acceder a su `i`-ésimo caracter.

3. Funciones y pasaje de parámetros

Ejercicio 3.1. Sin importar bibliotecas auxiliares, escribir un programa que:

(a) Dado $n \in \mathbb{N}$ calcule $\lfloor \sqrt{n} \rfloor$ (es decir, la parte entera de $\sqrt{n}$).

(b) Dados $n, m \in \mathbb{N}$ calcule $\lfloor \sqrt{n} \rfloor$ y $\lfloor \sqrt{m} \rfloor$.

¿Los programas de los puntos (a) y (b) se parecen? Lo ideal sería definir (y reusar) una función para el cálculo de la raíz cuadrada de un número.

Ejercicio 3.2. Escribir dos **funciones** *prod* y *pot* que dados $n, m \in \mathbb{N}$ calculen $n * m$ y n^m, respectivamente. No usar el operador $*$.

Ejercicio 3.3. Escribir un programa que:

(a) Dado $n \in \mathbb{N}$ calcule $n!$ (factorial de n).

(b) Dados $n, k \in \mathbb{N}$ (con $k \leq n$) calcule el combinatorio $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

(c) Dado $n \in \mathbb{N}$ imprima por pantalla los valores de $\binom{n}{i}$ para todo $i < n$.

En estos casos, nuevamente lo ideal es definir y reusar funciones.

Ejercicio 3.4. Sin usar el operador $*$ de strings, escribir un programa que:

(a) Dado $n \in \mathbb{N}$ imprima por pantalla una línea de n asteriscos.

(b) Dado $n \in \mathbb{N}$ imprima por pantalla un cuadrado de asteriscos de lado n.

(c) Dado $n \in \mathbb{N}$ imprima por pantalla un triángulo rectángulo de altura n.

Ejercicio 3.5. Sin ejecutarlo, determinar qué imprime por pantalla el siguiente código.

```
def f(x, y):
    x = 2 * x + y
    return x
x = 3
y = 7
y = f(y, x)
x = f(y, x)
print(x, y)
```

Ejercicio 3.6. Implementar el procedimiento `reversa` del siguiente programa.

```
text = "!ecneics retupmoc nioj"
print(reversa(text))
```

Ejercicio 3.7.

(a) Escribir un programa en Python que reciba un número n en base diez e imprima por pantalla su representación en base 2. Por ejemplo,

- entrada: `26` → salida: "`11010`"

(b) Repetir el punto (a), pero convierta n a base 16. Por ejemplo,

- entrada: `26` → salida: "`1A`"

4. Listas y archivos

Ejercicio 4.1. Escribir un programa que, dada una lista de strings, cuente la cantidad total de caracteres. Usar iteradores y no usar la función `len`.

Ejercicio 4.2. Dada la lista `x = list(range(0,10))`, determinar **a mano** qué devuelven las siguientes expresiones. Revisar evaluándolas en Python.

(a) x[3:7]

(b) x[3:]

(c) x[:3]

(d) x[:]

(e) x[3:-2]

(f) x[-7:7]

(g) x[-5:]

(h) x[:-5]

Ejercicio 4.3. Dada una secuencia de números enteros, llamamos *mesetas* a las subsecuencias de números iguales que aparecen en forma consecutiva. Por ejemplo, la secuencia [1,1,2,6,6,6,3,3] contiene las mesetas [1,1], [2], [6,6,6] y [3,3]. Escribir un programa en Python que determine el número y la longitud de la meseta más larga. Para la secuencia del ejemplo, el programa debería imprimir por pantalla "`número=6; longitud=3`". Puede definirse la lista en forma estática:

```
sec = [1,1,2,6,6,6,3,3]
```

Ejercicio 4.4. Implementar las funciones `getMin`, `getMax` y `computeMean` para que el siguiente programa tenga el comportamiento esperado. Las funciones no deben modificar el contenido de la lista.

```
1 valores = [23,4,67,32,13]
2 print("El minimo valor es: ", getMin(valores))
3 print("El maximo valor es: ", getMax(valores))
4 print("El promedio es: ", computeMean(valores))
```

Ejercicio 4.5. Escribir el siguiente programa; guardarlo con nombre "argumentos.py":

```
1 import sys
2 print("El nombre del script: ", sys.argv[0])
3 print("Cantidad de argumentos: ", len(sys.argv))
4 print("Los argumentos son: " , str(sys.argv))
```

(a) Ejecutar el programa mediante los siguientes comandos:

```
python argumentos.py
python argumentos.py hola
python argumentos.py hola 1 2 3 4
```

(b) ¿Qué conclusión puede sacar del contenido de `sys.argv`?

Ejercicio 4.6.

(a) Escribir una función que cuente la cantidad de palabras que tiene un texto pasado como parámetro.

(b) Escribir un programa que lea un archivo de texto y escriba un nuevo archivo indicando la cantidad de palabras de cada línea. Ejemplo:

```
ARCHIVO DE ENTRADA                          ARCHIVO DE SALIDA
------------------                          -----------------
Lorem ipsum dolor sit amet,                  5
  consectetur adipiscing elit.  Integer      4
                                             0
nec odio...                                  2
```

Ejercicio 4.7. La función `range()` toma uno, dos o tres parámetros de entrada: `range(stop)`, `range(start, stop)`, o bien `range(start, stop, step)`. Investigar el funcionamiento de cada variante. ¿Qué parámetros deben pasarse para generar la lista `[27, 44, 61, 78, 95]` con esta función?

Copia vs. referencia

Ejercicio 4.8. ¿Qué valor se imprime en la última instrucción del siguiente programa?

```
1 a = [1,2]
2 b = a
3 a.append(3)
4 print(len(b))
```

¿Por qué no imprime 2, si luego de ejecutar la segunda instrucción, `b` tenía 2 elementos y luego no se modifica? Comparar con el ejercicio 1.6. ¿Qué se imprime si reemplazamos la segunda instrucción por `b = list(a)`?

Ejercicio 4.9. Sin ejecutarlo, determinar qué imprime por pantalla el siguiente código.

```
1 def despedir(x):
2   x.append('chau')
3   print(len(x))
4
5 a = ['hola', 'mundo']
6 despedir(a)
7 print(a)
8 despedir(list(a))
9 print(a)
```

Ejercicio 4.10. Sin ejecutarlo, determinar qué imprime por pantalla el siguiente código.

```
1 def suma_problematica(lista):
2   n = 0
3   for x in lista:
4     n += x
5   lista.clear()
```

```
6       return n
7
8   a = [1,2,3,4]
9   print(suma_problematica(a))
10  print(suma_problematica(a))
```

¿Qué cambia si reemplazamos la línea 5 por `lista = []`? ¿Por qué no es lo mismo?

Listas por comprensión

Ejercicio 4.11. Determinar (primero sin la computadora; luego validar en `ipython`) el valor de estas expresiones, donde `txt = 'este es un texto muy corto de prueba'`.

(a) `[x for x in range(0, 10) if x%3==1]`

(b) `[(x if x%2==0 else -111) for x in range(0, 10)]`

(c) `[x for x in txt.split(' ') if len(x)>3]`

(d) `' '.join([x+' '+x for x in txt.split()])`

(e) `''.join([x for x in txt if x not in 'aeiou'])`

(f) `[x+y for x in 'ab' for y in '123']`

Ejercicio 4.12. Escribir expresiones con listas por comprensión que devuelvan estos valores, tomando nuevamente como punto de partida `range(0, 10)` o bien la variable `txt` definida en el ejercicio anterior.

(a) `[10, 50, 90]`

(b) `['par', 'non', 'par', 'non', 'par', 'non', 'par', 'non', 'par', 'non']`

(c) `'_st_ _s _n t_xt_ m_y c_rt_ d_ pr__b_'`

(d) `'EstE Es Un tExtO mUy cOrtO dE prUEbA'`

(e) `[4, 2, 2, 5, 3, 5, 2, 6]`

(f) `['a1', 'b1', 'a2', 'b2', 'a3', 'b3']`

5. Tipos de Datos

Ejercicio 5.1.

1. El **índice Jaccard** es una medida de similitud entre conjuntos. Se define como

$$\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

   Jaccard(A, B) se mueve entre 0 (cuando A y B son totalmente disímiles) y 1 (cuando son iguales). Escribir una función en Python que compute el índice Jaccard. Por ejemplo, `jaccard(set([1,2,3]), set([3,4]))` debe devolver 0.25.

2. Se denomina ***k*-shingles** a las subsecuencias de caracteres de longitud k en un string. Escribir una función en Python que, dados un string s y un entero k, devuelva el conjunto de k-shingles de s. Por ejemplo, `k_shingles('hola mundo', k=5)` devuelve el conjunto formado por `'hola '`, `'ola m'`, `'la mu'`, `'a mun'`, `' mund'` y `'mundo'`.

3. Escribir un programa en Python que, dados dos textos breves, extraiga sus k-shingles y compute la similitud Jaccard entre ambos conjuntos. Esta técnica se usa frecuentemente en la detección automática de plagio en textos.[1]

Ejercicio 5.2. Escribir un programa que deletree palabras. Por ejemplo, dado `"aljibe"` debe devolver `"a, ele, jota, i, be, e."` Usar un diccionario para representar las reglas. Tener en cuenta que muchas letras se pronuncian parecido (ej.: be, ce, de, ge, ...; efe, eme, ene, ...) para automatizar la carga lo máximo posible, en lugar de escribir todas las reglas a mano.

Ejercicio 5.3.

a) Definir el tipo Fecha con las siguientes operaciones, e implementarlo en Python (no vale usar el tipo `date`, ni similares).

   - `Fecha(d, m, a)`: Construye un nuevo elemento de tipo Fecha.
   - `f.siguiente()`: Devuelve la fecha siguiente a la fecha dada. (Ej: al 31/12/1999 le sigue el 1/1/2000.)
   - `f1 <f2`: Devuelve `True` si la fecha f1 es anterior a la fecha f2, y `False` en caso contrario.

b) Usar el tipo Fecha para resolver los siguientes problemas:

   (I) Contar la cantidad de días entre dos fechas f1 y f2 usando este código:

```
cantidad = 0
while f1 < f2:
  cantidad = cantidad + 1
  f1 = f1.siguiente()
```

   (II) Contar la cantidad de días "29 de febrero" entre dos fechas dadas.

Ejercicio 5.4. Implementar el tipo Stack (pila) con las siguientes operaciones. Usar una lista como estructura de representación.

- `p = Stack()`: Crea una pila vacía.

[1] Lectura recomendada: *Mining of Massive Datasets* (`http://www.mmds.org`), sección 3.1.

- `p.push(x)`: Inserta el elemento x sobre el tope de la pila p.
- `p.empty()`: Dice si la pila p está vacía.
- `p.top()`: Devuelve el elemento del tope de p. Precondición: `not p.empty()`
- `p.pop()`: Borra el elemento del tope de p. Precondición: `not p.empty()`

Ejercicio 5.5. Escribir un algoritmo que determine si un string dado está *balanceado* con respecto a los caracteres { }, [ ], ( ). Ejemplos: "{a(b)x[()]}" está balanceado; "}", "a(b))" y "[[})" no están balanceados. Sugerencia: Usar el tipo Stack().

Ejercicio 5.6. Implementar el tipo Queue (cola) con las siguientes operaciones. Usar una lista como estructura de representación.

- `c = Queue()`: Crea una cola vacía.
- `c.enqueue(x)`: Inserta el elemento x al final de la cola c.
- `c.empty()`: Dice si la cola c está vacía.
- `c.first()`: Devuelve el elemento que está primero en c. Precondición: `not c.empty()`
- `c.dequeue()`: Borra el primer elemento de c. Precondición: `not c.empty()`

Ejercicio 5.7. Implementar el tipo AtenciónAlCliente, que consiste de dos colas con distinta prioridad: VIP y Común. Si en la cola VIP hay clientes, se los atiende primero (en orden de llegada); si no, se atiende a los clientes de la cola Común. El tipo debe ofrecer las siguientes operaciones:

- `a = Atención()`: Crea un nuevo sistema de atención al cliente.
- `a.llega(c, p)`: Llega el cliente c (un `string`), con prioridad `'vip'` o `'común'`.
- `a.vacía()`: Devuelve `True` si ambas colas están vacías, y `False` en caso contrario.
- `a.atender()`: Devuelve el cliente al que le toca ser atendido, y lo saca de la cola correspondiente. Precondición: `not a.vacía()`

Ejemplo:

```
a = AtenciónAlCliente()
a.llega("María", "común")
print(a.vacía())      #  imprime "False"
a.llega("Juan", "común")
print(a.atender())    #  imprime "María"
a.llega("Pedro", "vip")
print(a.atender())    #  imprime "Pedro"
print(a.atender())    #  imprime "Juan"
print(a.vacía())      #  imprime "True"
```

Ejercicio 5.8. (Opcional)

a) Definir el tipo Tatetí, que provea operaciones para crear un juego, determinar de quién es el turno, hacer una jugada (cruz o círculo), determinar si el juego terminó, y (en caso afirmativo) quién fue el ganador o si hubo empate.

b) Implementar el tipo en Python.

c) (Opcional) Escribir un programa en Python que (usando el tipo Tatetí) permita al usuario jugar interactivamente contra la computadora, mostrando por pantalla el tablero y preguntando la siguiente movida. Las jugadas de la computadora pueden realizarse al azar.

6. Complejidad algorítmica, búsqueda y ordenamiento

Ejercicio 6.1. Estimar el orden de complejidad temporal (en peor caso) de los algoritmos realizados para los ejercicios 3.1(a), 3.2 y 3.3(a).

Ejercicio 6.2. Para este ejercicio, adaptar los algoritmos de búsqueda vistos en clase.

(a) Escribir un algoritmo que, dados una lista A y un entero x, retorne todos los elementos de A menores o iguales que x. Estimar el orden de complejidad temporal (en peor caso) del algoritmo.

(b) Igual al punto anterior, pero sabiendo que la lista está ordenada.

Ejercicio 6.3.

(a) El algoritmo BUBBLESORT consiste en recorrer la lista A de izquierda a derecha, dejando en cada paso ordenados los elementos $A[i]$ y $A[i+1]$; este proceso debe repetirse $len(A)$ veces. Implementar el algoritmo en Python. ¿Por qué funciona? Describir el estado de la lista A al finalizar cada recorrida del ciclo interno.

(b) Mostrar que BUBBLESORT tienen complejidad $O(len(A)^2)$.

Ejercicio 6.4. Escribir un algoritmo que resuelva cada uno de los siguientes problemas con el orden de complejidad temporal indicado. Mostrar que el algoritmo hallado tiene el orden indicado.

1. Calcular la media de una lista de enteros. $O(len(A))$
2. Calcular la mediana de una lista de enteros. $O(len(A)^2)$
3. Determinar, dado un n natural, si n es (o no) primo. $O(\sqrt{n})$
4. Dado un n natural, obtener su representación binaria (como una lista de 1s y 0s). $O(\log n)$
5. Dada una lista de 1s y 0s representando un número en base binaria, obtener el número correspondiente en base decimal. $O(len(A))$

(Opcional) ¿Considera que en alguno de los casos anteriores la complejidad del algoritmo propuesto es *óptima* (en términos de orden de complejidad)?

Ejercicio 6.5. (Opcional) Dadas dos clases de complejidad $O(f)$ y $O(g)$, decimos que $O(f) \leq O(g)$ si y sólo si para toda función de complejidad $T \in O(f)$ sucede que $T \in O(g)$.

1. ¿Qué significa, intuitivamente, $O(f) \leq O(g)$? ¿Qué se puede concluir cuando, simultáneamente, tenemos $O(f) \leq O(g)$ y $O(g) \leq O(f)$?
2. ¿Cómo ordena $\leq$ las siguientes clases de complejidad?

- $O(1)$
- $O(\sqrt{n})$
- $O(\sqrt{2})$
- $O(\log n^2)$
- $O(\log n!)$
- $O(n+1)$
- $O(1/n)$
- $O(\log n)$
- $O(\log \log n)$
- $O(2^n)$
- $O(n^2)$
- $O(n^n)$
- $O(\log^2 n)$
- $O(n!)$
- $O(n \log n)$

7. Verificación de Programas

Ejercicio 7.1. Diseñar y ejecutar testing de unidad para las funciones de los ejercicios 3.1(a), 3.2 y 3.3(a).

Ejercicio 7.2. Diseñar y ejecutar testing de unidad para todas las funciones públicas de los tipos definidos en los ejercicios 5.3, 5.4 y 5.6.

Ejercicio 7.3. Diseñar y ejecutar testing de unidad para las funciones de los ejercicios 5.1, 5.2. 5.5 y 5.7.

Ejercicio 7.4. El archivo `debugging.py` contiene varias funciones, cada una con una descripción de la funcionalidad deseada. Sin embargo, todas tienen al menos un error. Se pide *debuggear* y corregir cada función con el siguiente procedimiento:

1. Definir casos de test antes de mirar o ejecutar el código.
2. Ejecutar los casos de test, e identificar al menos uno para el cual la función falle.
3. Agregar `print`s en el código para inspeccionar las variables de la función.
4. Describir cómo debería ser la evolución esperada de las variables del programa.
5. Ejecutar la función con algún caso de test problemático, haciendo un seguimiento manual de las variables.

Luego de cada cambio, volver a correr los casos de test.