Curso de Nivelación de Algoritmos
Guía de Ejercicios

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 y averiguar su tipo con la operación type().

(a) 1 + 1 => Int + int => int

(b) ’ho’ + ”la” Concatenacion de strings => string

(c) (1>0) or (not (’a’<’b’)) => Resultado boolean con literal False (Expresión booleana)

(d) len(’hola’) + 6 => Resultado int => 10

(e) (5.6 >2.0) and (len(’hola’)<2) True and False => False (Expresión booleana)

(f) 1 + 1.0 => 1.0 el entero se convierte al timo más general Float.

(g) 11 / 2 => 5.5 (El resultado es un Float)

(h) 11 // 2 => 5 (El resultado es un Int)

(i) 1 / 0 => Error de división por cero.

(j) 123 + ’123’ => Deveria convertir el entero 123 a string => str(123) + ‘123’ => ‘123123’(String)

(k) 1 + int(’123’) => 124 (Int)

(l) str(123) + ’123’ => ‘123123’ (String)

Comparar el resultado de evaluar las expresiones (g) y (h).
Re:

- (g): Se aumenta la precisión de 11 y 2 a Float para tener un resultado con decimales.
- (h): Redondea hacia abajo siempre. 1.8, 1.9 o 1.001 se redondea a 1.

¿A qué se deben los errores de las expresiones (i) y (j)?

(i) División por cero.
(j) el operador ‘+’ no se comporta de igual forma para los enteros y los string. Para enterizo suma los valores, para los string los concatena. Para mi lo mejor es usar interpolación de strings.

Ej:

```
number = 123
message = f'Number: {number}'
pruint(message)
```

¿Qué ocurrió exactamente en las expresiones (k) y (l)?

(k) Se convierte el string '123' a enterio y se suma con otro enterio.
(l) Idem pero lo contrario: Se convierte el entero 123 a string y se concatena con el string '123'

Ejercicio 1.2.
(a) Escribir el siguiente programa en un archivo holamundo.py:

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

Ejecutar el programa y observar su salida 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.

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

Error ya que no se encuentra el símbolo de cierre del string.

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

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

\n es nueva linea.
\t es tabulacion.

=> resultado 3 lines y la última una tabulación entre chau y mindo

(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
""")
```

```
“””linea1
linea2
linea n
“”
```

Es un string multilínea. Es la forma de escribir un string en varias líneas. En el punto (b) podríamos haber hecho lo siguiente:

```
print(“””hola
mundo”””)
```

Ejercicio 1.3.
En el ejercicio anterior apareció el carácter \. Se conoce a la barra invertida como carácter de escape, y su función dentro de una cadena de carácteres es darle un significado especial al carácter que figura inmediatamente a continuación. Por ejemplo, \n se usa como carácter de nueva línea. Escribir las siguientes instrucciones en la consola:

```
print("\"\n")
```
Re: Imprime “ y luego nueva línea

```
print("""Cuidado con las "comillas".""")
```
Re:

- Entre “””...””” no hay que escapar “ o ‘
- Entre “...”
    - hay que escapar las comillas dobles \” Para que python no las confunda con el fin del string.
    - Se puede usar comillas simples sin problemas ya que no es símbolo de apertura y cierre.
- Entre ‘...’ es parecido al anterior:
    - Se puede usar “
    - Hay que escapar las comillas simples \’.

```
print('Tengamos "Mucho cuidado" por favor.')
```
Re: Se imprime son problemas.

Ejercicio 1.4.
¿Qué valor se imprime en la última instrucción del siguiente programa?

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

Re: Imprime 1.

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

Re: Por que los tipos básicos siempre se **pasan por valor.** Si fueran instancias de clases (Objetos) si se pasarían por referencia.

Pasar por valor: Se hace una copia del valor. Python pasa por valor todos los tipos básicos: Int, Float, String, etc…

Ejercicio 1.5.
¿Qué se imprime en las instrucciones 2 y 4 del siguiente programa?

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

Re: Imprime:

- Int
- String

Los tipos de cada valor que tome la variable a.

Entonces, ¿cuál es el tipo de la variable en este programa?

Re: Python es un lenguaje no tipado. Las variables no se pre-definen para un tipo dado, pueden:

- Guardar cualquier tipo de dato básico.
- Referenciar instancias de distintas clases(Objetos).

En general no se cambia el tipo de dato de una variable, pero sí es común referenciar objetos(Instancias de clases) que comparten una misma interfaz para usar polimorfismo.

Ejercicio 1.6.
Escribir un programa saludador.py que se comporte de la siguiente manera:

```
$ python saludador.py Juan
Hola Juan
$ python saludador.py Alicia
Hola Alicia1
```

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

```
#!/bin/python
import sys

if len(sys.argv) < 2:
   print("Name argument is required!")
   exit(1)

name = sys.argv[1]
print(f'Hello {name}')
```

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 milla es 1,61 km).
La tabla debe contener solo la parte entera de la conversión, empezar en 0 mi, terminar en 100 mi e imprimir los valores en km tomando intervalos de 10 mi. 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.
Re: Ver `ejercicio-2.1.py`

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.
Re: Ver `ejercicio-2.2.a.py`

(b) Repetir el punto (a), pero para los primeros n números impares.
Re: Ver `ejercicio-2.2.b.py`

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:

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

Re: Ver `ejercicio-2.3.py`

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

(a) **Re:** Ver `ejercicio-2.4.a.py`
(a) **Re:** Ver `ejercicio-2.4.b.py`
(c) **Re:** Ver `ejercicio-2.4.c.py`
(d) **Re:** Ver `ejercicio-2.4.d.py`

Ejercicio 2.5.
Escribir un programa en Python que determine si un número entero n dado es primo, es decir, que determine si n es divisible solo por n, −n, 1 y −1.
Re: Ver `ejercicio-2.5.py`

Ejercicio 2.6.
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 10.000 números naturales (deben encontrarse exactamente cuatro números perfectos).
Re: Ver `ejercicio-2.6.py`

3. Funciones y pasaje de parámetros

Ejercicio 3.1.
Escribir dos funciones prod y pot que dados n, m naturades calculen n*m y n elevado a la m, respectivamente. No usar el operador * de Python.
Re: Ver `ejercicio-3.1.a.py`
Re: Ver `ejercicio-3.1.b.py`

Ejercicio 3.2.
Escribir un programa que:
(a) Dado n ∈ N calcule n! (factorial de n).
Re: Ver `ejercicio-2.4.a.py` (Ya se pregunto en la parte 2).

(b) Dados n, k ∈ N (con k ≤ n) calcule el combinatorio:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Re: Ver `ejercicio-3.2.b.py`

(c) Dado n ∈ N imprima por pantalla los valores de $\binom{n}{i}$ para todo i < n.
Re: Ver `ejercicio-3.2.c.py`

Para todos estos casos lo ideal es definir y reusar funciones.

Ejercicio 3.3.
Sin usar el operador $*$ de strings, escribir un programa que:
(a) Dado $n \in \mathbb{N}$ imprima por pantalla una línea de n asteriscos.
Re: Ver `ejercicio-3.2.a.py`

(b) Dado $n \in \mathbb{N}$ imprima por pantalla un cuadrado de asteriscos de lado n.
Re: Ver `ejercicio-3.2.b.py`

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

Ejercicio 3.4.
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)
```

Re:
```
y = f(7, 3) => 2 * 7 + 3
Y => 17
x = f(17, 3) => 2 * 17 + 3
x => 37
=> Imprime 37 17.
```

Ejercicio 3.5.
Escribir en Python la función palindromo que, dada una palabra, determine si se trata de un palíndromo (palabra capicúa). Por ejemplo:

- palindromo(anilina) → True
- palindromo(harina) → False

Re: Ver `ejercicio-3.5.py`

Ejercicio 3.6.
Implementar la función reversa del siguiente programa.

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

Re: Hay que usar la función reserve implementada en el ejercicio anterior. Ver `ejercicio-3.6.py`

4. Listas

Ejercicio 4.1.
Escribir un programa que, dada una lista de strings, cuente la cantidad total de caracteres.

Re: Ver `ejercicio-4.1.a.py o ejercicio-4.1.b.py`

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]
Re: Desde el índice 3 al 6 inclusive. En este caso los índices coinciden con el contenido de cada posición de la lista así que sería 3, 4, 5 y 6.

(c) x[:3]
Re: Desde el índice 0 al 2 inclusive. Serían los valores: 0, 1 y 2.

(e) x[3:-2]
Re: Comienza en 3 y finaliza en 9 - 2 => 7. Seria: 3, 4, 5, 6 y 7.

(g) x[-5:]
Re: Comienza desde -5 hasta el final de la lista. Seria: 5, 6, 7,8 y 9.

(b) x[3:]
Re: Desde la posición 3 en adelante: Sería: 3, 4, 5, 6, 7, 8 y 9.

(d) x[:]
Re: Todo desde el principio al final.

(f) x[-7:7]
Re: Comienza el la posición 3 (-7 desde atrás hacia adelante) hasta la posición 6 inclusive. Seria: 3, 4, 5 y 6.

(h) x[:-5]
Re: Comienza desde 0 y termina en la posición 4 inclusive. Sería: 0, 1, 2, 3 y 4.

Conclusiones

- El índice de comienzo siempre incluye la posición a la que se refiere.
- El índice de fin siempre se refiere a la posición anterior desde comienzo de la lista, sea un índice positivo o no.

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 deberá imprimir por pantalla "`número=6; longitud=3`". Puede definirse la lista en una variable dentro del código:

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

Re: Es el ejercicio 7 del examen 2017. Ver ./examen/2017/`ejercicio-7.1.py`

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.

```
valores = [23,4,67,32,13]
print("El mínimo valor es: ", getMin(valores))
print("El máximo valor es: ", getMax(valores))
print("El promedio es: ", computeMean(valores))
```

Re: Ver `ejercicio-4.4.py`

Ejercicio 4.5.
¿Qué valor se imprime en la última instrucción del siguiente programa?

```
a = [1,2]
b = a
a.append(3)
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 1.4. ¿Qué se imprime si reemplazamos la segunda instrucción por b = list(a)?

Re:
Entonces;

```
a = [1,2]
b = a → b referencia a a.
a.append(3) a → a = [1, 2, 3]
print(len(b)) → Como b referencia a a entonces len(a) == len(b).

=> Imprime 3.
```

Ejercicio 4.6.
Sin ejecutarlo, determinar qué imprime por pantalla el siguiente código.

```
def despedir(x):
  x.append('chau')
  print(len(x))

a = ['hola', 'mundo']
despedir(a)
print(a)
despedir(list(a))
print(a)
```

Re:

1. **a = ['hola', 'mundo']**
2. **despedir(a)**
    - **a** = ['hola', 'mundo', 'chau'] e imprime 3.
    - Se pasa la referencia de **a** como argumento de la función **despedir.**
    - A es un objeto no es un valor, ojo!.
3. **print(a)**
    - Imprime ['hola', 'mundo', 'chau']
4. **despedir(list(a))**
    - Hace una copia de **a** e imprime 4.
    - **Nota:** Se pasa una referencia a un nuevo objeto copia de objeto referencia por la variable **a.**
5. **print(a)**
    - Imprime **a** otra vez.

Salida:

```
3
['hola', 'mundo', 'chau']
4
['hola', 'mundo', 'chau']
```

5. Complejidad algorítmica y búsqueda

Ejercicio 5.1.
Estimar el orden de complejidad temporal (en peor caso) de los algoritmos realizados para los ejercicios 3.1 y 3.2(a).

3.1.a

```
import sys (1)

def multiply(a, b): (8 + 3b)
   mult = [a for _ in range(0, b)] (5 + 3b)
```

- 1: Leo el valor de la variable b.
- 1: Se crea el rango.
- 1: La abstracción toma el rango.
- 1: Se crea la lista donde se acumulan los resultados de cada iteración de bucle.
- 3b
    - 1: Se lee una posición del rango, más allá de que se use o no.
    - 1: Se lee el valor de la variable a.
    - 1: Se agrega el valor a la lista resultado.
- 1: Se asigna la referencia de la nueva lista con todos los valores a la variable mult.

```
   return sum(mult) (3)
```

- 1: Se lee la referencia de la variable mult.
- 2: se suman todos los elementos de la lista.
- 3: Se asigna o retorna el valor resultado.

```
a = int(sys.argv[1]) (3)
b = int(sys.argv[2]) (3)
result = multiply(a, b) (9 + 3b)
print(f'{a} * {b} => {result}') (4)
```

Total: 20 + 3b → O(n) (Orden lineal)

3.2.

```
import sys (1)

def fact(number): (4 + 4(n-1))
   if number == 1: (2)
       return number (2)
   else:
       return number * fact(number - 1) (4 + 4(n-2))
```

- 1: Leer la primera vez number.
- 1: Leer la variable number.
- 1: La operación de resta.
- 1: La asignación del resultado de la resta al argumento de fact

```
def nCr(n, r):
  a = fact(n) (4 + 4n) + 1
  b = fact(r) (4 + 4r) + 1
  c = fact(n - r) (4 + 4(n - r)) + 1
  return a / (b * c) (3 + 3)
```

Total = 7 + 4 + 4n + 4 + 4r + 4 + 4(n - r)
Total = 19 + 4n + 4r + 4n - 4r
Total = 19 + 8n

```
n = int(sys.argv[1]) (3)
r = int(sys.argv[2]) (3)
ncr = nCr(n, r) (19 + 8n) + 1
print(f'{n}C{r} => {ncr}') (5)
```

Total = 26 + 8n → Order Lineal O(n)

Ejercicio 5.2.
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.
Re: Ver `ejercicio-5.2.1.py`

(b) Igual al punto anterior, pero sabiendo que la lista está ordenada.
Re: Ver `ejercicio-5.2.2.py`

Ejercicio 5.3.
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. $O(len(A))$
   Re: Ver `ejercicio-5.3.1.py`
2. Calcular la mediana de una lista de enteros. $O(len(A)^2)$
   Re: Ver `ejercicio-5.3.2.py`
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)?

6.Recursión

Ejercicio 6.1.
Programar las siguientes funciones recursivas en Python:
(a) pot_dos(n), que dado un número $n \in N$ calcúle 2 elevado a la n . Ayuda: 2 a la 0 = 1.
(b) pot_a(a, n), que dados a, $n \in N$ calcule a elevado a la n .

Ejercicio 6.2.
La sucesión de Fibonacci es una secuencia de números en la cual los dos primeros términos son 1 y los siguientes se definen como la suma de los dos anteriores. A continuación, se observan los primeros diez términos de la sucesión:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, . . .

Programar una función recursiva en Python fibonacci(n) que dado un número $n \in N$, devuelva el n-ésimo término de la sucesión de Fibonacci.

Ejercicio 6.3.
Programar una función recursiva en Python es_par(n) que determine si un número $n \in N$ es par (i.e., que devuelva True si es par y False en caso contrario).

Ejercicio 6.4.
Programar una función recursiva en Python productoria(l) que dada una lista no vacía de enteros l devuelva el resultado de multiplicar todos los números de l.

Ejercicio 6.5.
Programar una función recursiva en Python cantidad_ocurrencias(n,l) que dada una lista de enteros l y un número n, devuelva la cantidad de ocurrencias de n en l.