

# 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"` Concatenación 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 tipo 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'` => Debería 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 enteros suma los valores, para los string los concatena. Para mí lo mejor es usar interpolación de strings.

Ej:

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

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

(k) Se convierte el string '123' a entero y se suma con otro entero.

(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 mundo

(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 caracteres 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\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 Alicia
```

**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  $n$ ,  $m$  naturales calculen  $n \cdot m$  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  $n \in \mathbb{N}$  calcule  $n!$  (factorial de  $n$ ).

**Re:** Ver `ejercicio-2.4.a.py` (Ya se preguntó en la parte 2).

(b) Dados  $n, k \in \mathbb{N}$  (con  $k \leq n$ ) calcule el combinatorio:

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

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

(c) Dado  $n \in \mathbb{N}$  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`

### Ejercicio 3.6.

Implementar la función `reversa` del siguiente programa.

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