1. El lenguaje Python, además de ser usado para interpretar y ejecutar programas ya escritos (con extensión `.py`), puede ser usado en forma *interactiva*, es decir, ingresando comandos que serán ejecutados inmediatamente en la consola. Para esto, realice las siguientes tareas:
   (a) Abra una consola y ejecute el intérprete de Python (es decir, ejecute el comando `python`, sin indicar un arhivo `.py`). ¿Qué versión de Python está instalada?
   (b) Ejecute los siguientes comandos en forma consecutiva:

```
x = 1
y = 2
print(x,y)
print("El valor de x es ",x," y el valor de y es ",y)
```

   (c) ejecute `quit()` para salir del intérprete interactivo y volver a la consola (Bash).
   (d) Vuelva a ingresar a una sesión interactiva de Python y ahora ejecute:

```
sx = str(x)
type(sx)
```

   ¿Qué tipo de variable es `sx`? Entonces, ¿qué hace la función `str()`?
   (e) Ahora ejecute:

```
mensaje = "El valor de x es " + str(x) + " y el valor de y es " + str(y)
print(mensaje)
```

   ¿Qué diferencia observa en el resultado?
   (f) Conozca la función `len()`, para esto ejecute:

```
n = len(mensaje)
print(n)
type(n)
```

   ¿Qué valor entrega la función `len()` aplicada a un string[1]?, ¿Qué tipo de variable suministra? (pruebe aplicándola a otros strings).

2. Existen diversas operaciones definidas entre distintos tipos de variables. Para aprender cómo funcionan algunas de ellas defina primero las siguientes variables y verifique su tipo:

```
a = 3.14
b = 2
c = 5
d = 6+2j
e = "hola "
f = "mechones"
g = True
```

[1] "String" es el nombre usado comúnmente para una *cadena* de caracteres alfanuméricos.

A continuación imprima el valor y el tipo del resultado de las siguientes operaciones: `a+b, a+d, a+e, b+c, b+d, b+e, f+e, e+f, a*b, a*d, a*e, b*c, b*d, c*e, e*f, a**b, a**d, a**e, b**c, e**a, e**b, e**f, a/b, a/d, a/e, b/c, b/d, b/e, c/b, d/a, d/b, e/a, e/b, e/f, a*g, b*g, not(g), g and False, g and True, g or False, g or True`. ¿Cuáles de estas operaciones no están definidas?

- ¿Qué pasó en los casos `b/c` y `c/b`?. Busque en las referencias sugeridas la explicación de este comportamiento.
- También existen operaciones que transforman el tipo de variable. Por ejemplo, como continuación del ejercicio anterior, calcule y verifique el tipo de las siguientes operaciones: `int(a), float(b), d.real, d.imag, a==b, a>b`.
- Cree un programa Python llamado `test01.py` e incluya como primeras líneas el siguiente código:

```
print("Resolveremos la ecuacion a*x**2 + b*x + c = 0")
a = float(input("Valor de a = "))
b = float(input("Valor de b = "))
c = float(input("Valor de c = "))
```

Este pequeño programa Python, al ser ejecutado con el comando `python test.py`, pregunta al usuario por los valores de las variables a, b y c, que son asignadas como valores decimales (float). Ahora modifique el programa para que además *calcule e imprima* las dos soluciones de la ecuación cuadrática, es decir, los valores

$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{1}$$

Para calcular la raiz cuadrada involucrada eleve el valor correspondiente la potencia 0.5, es decir, use el hecho que $\sqrt{\alpha} = \alpha^{0.5}$.

3. Los caracteres individuales que forman una cadena alfanumérica (string) pueden ser accesados usando el número del *índice* correspondiente. En Python **el valor de los índices siempre comienza en 0**, luego 1, 2, etc. Para ilustrar esto, abra un intérprete Python y ejecute los siguientes comandos:

```
x = "Hola estudiantes de primer semestre"
print(x[0])
print(x[1])
print(x[2])
print(x[3])
```

4. Como comprobó anteriormente la función `len()` entrega el largo del string, es decir, el número de caracteres que contiene. Por lo tanto

```
print(x[len(x)-1])
```

imprime el último caracter del string, cuyo índice es `len(x)-1`, debido que el índice del primer caracter es 0. El mismo resultado, puede ser conseguido usando

```
print(x[-1])
```

Similarmente,

```
print(x[-2])
```

imprime el penúltimo caracter, y así sucesivamente. Por ejemplo, ejecute

```
x[-3] == x[len(x)-3]
```

para comprobar que se refieren al mismo caracter.

5. También es posible acceder a un subconjunto de caracteres del string usando, en nuestro caso, `x[inicio:fin:paso]`, donde `inicio` y `fin` son los índices de los caracteres iniciales y finales y `paso` es un entero que define el paso. Si `paso` no es ingresado, el intérprete considera que `paso = 1` . Por ejemplo, ejecute y verifique qué hacen los siguientes comandos

```
print(x[0:4:1])
print(x[0:4])
print(x[5:16])
print(x[1:20:2])
```

Note que el caracter correspondiente al índice `fin` NO es desplegado. En lenguaje matemático podríamos decir que `x[inicio:fin]` suministra los caracteres de `x` con índices en el intervalo desde `inicio` *cerrado* hasta `fin` *abierto*.

6. Además, si no se especifica `inicio` se asume el valor 0 (inicio del string) y si no se especifica `fin` se asume el valor `len(x)` (fin del string). Verifique esto ejecutando:

```
print(x[:4])
print(x[5:])
print(x[5:-3])
print(x[::-1])
```

7. **Bonus track**:

   ¿Qué hace cada uno de los siguientes comandos?, ¿Modifican el valor de `x`?

```
x.upper()
x.replace("a","e")
x.find("p")
x.find("semestre")
```

8. Otro concepto muy importante en Python es el de *listas*. Las listas son similares a las cadenas, excepto que cada elemento puede ser de un tipo diferente. La sintaxis para crear listas en Python es [..., ..., ...]. Por ejemplo, ejecute:

```
lista = [1, "hola", 1.0, 1-1j, True]
type(lista)
print(lista)
```

Como puede ver, la variable `lista` es un nuevo tipo de objeto: 'list'. En este caso, es una lista cuyos elementos son un entero, un string, un float, un complejo, y un booleano. Para verificar esto, imprima el valor y el tipo de cada elemento de la lista. Por ejemplo,

```
print(lista[0],type(lista[0]))
print(lista[1],type(lista[1]))
```

Este ejemplo también muestra que los índices de cada elemento de la lista son numerados de la misma manera que en un string:

```
print(lista[0:3])
print(lista[::2])
```

9. Los elementos de una lista pueden tener cualquier tipo reconocido por Python, por ejemplo, pueden ser otra lista!:

```
superlista = ["cool",lista]
print(superlista)
```

Imprima el valor y el tipo de cada elementos de esta lista. ¿Cuántos elementos tiene la lista `superlista`? (respuesta, use la función `len()`).

10. Existen diversas funciones en Python que crean listas. La función `list()` crea una lista, por ejemplo, a partir de un string. Usando el string `x` definido anteriormente, ejecute

```
y = list(x)
print(y)
print(type(y))
```

11. Otra función que crea listas útiles, esta vez de números *enteros*, es `range(inicio,fin,paso)`, que crea una lista de valores desde `inicio` (cerrado) hasta `fin` (abierto!!), con paso `paso`. Ejecute,

```
z = list(range(2,26,3))
print(z)
```

12. ¿Qué hacen los siguientes comandos?, ¿Modifican el valor de `x` y/o `lista`?

```
x.split(" ")
x.split("e")
lista.append("chao")
lista.insert(2,"cool")
```