UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FCFM
FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Inteligencia Artificial

Tarea 1

Docente: Juan Pablo Rosas Baldazo

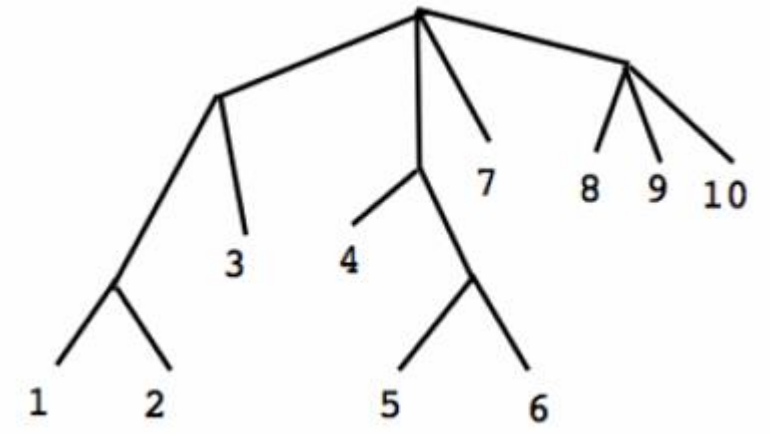
1735839 Bryan Aguirre Tudon

Índice

Instrucciones

Programar las siguientes funciones:

1- Función que calcule el cubo de un numero entero. **Ejemplo:** cubo(2) devuelve el valor de 8.
2- Función que calcule el valor factorial de un numero entero positivo. **Ejemplo:** factorial(4) devuelve el valor de 24.
3- Función que cuente cuantas veces se repite el patrón en una cadena de caracteres incluyendo si se superponen. **Ejemplo:** cuenta_patron((‘ab’),(‘abcebabf’)) debería regresar el valor de 2, y cuenta_patron((‘aba’),(‘gabababa’)) debería regresar el valor de 3.
4- Función que devuelva la parte de un árbol de acuerdo con su índice.



Ejemplo: El árbol anterior se puede representar en Python por [**[[1, 2], 3]**, **[4, [5, 6]]**, **[7]**, **[8, 9, 10]**]. Por lo que, para seleccionar el número nueve de esa estructura deberíamos poder hacerlo de la siguiente forma arbol_ref(tree, (3,1)), donde ‘tree’ representa la tupla con los datos del árbol y (3,1) la rama y el lugar que ocupa ese valor en esa rama. Tomemos en cuenta que estamos tomando índices iniciando en 0. Otro ejemplo sería obtener el 6, por lo que la forma de hacerlo sería arbol_ref(tree, (1,1,1)). También hay que tener en cuenta que no es necesario que siempre devuelva una hoja, puede devolver una parte del árbol, ejemplo: arbol_ref(tree, (0,)) debería devolver ((1,2),3).

5- Función que desarrolle una expresión algebraica. **Ejemplo:** (2 * (x + 1) * (y + 3)), debería devolver: ((2 * x * y) + (2 * x * 3) + (2 * 1 * y) + (2 * 1 * 3)) o su versión simplificada ((2 * x * y) + (6 * x) + (2 * y) + 6)).

Función 1

Función que calcule el cubo de un numero entero.

Código

```python
def cubo(num):
    return num**3

numero = int(input("Numero: "))

print("El cubo de: ",numero," es de: ", cubo(numero))
```

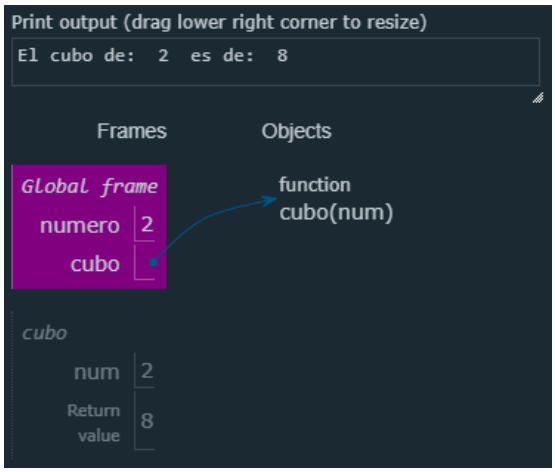
Explicación

Primero definimos lo que es la función **cubo()** con el argumento **num** con el cual vamos a trabajar con él, le aplicamos el exponente con el operador **** y colocamos el **3** para indicar que sea a la tercera potencia que tiene que elevar este número.

Definimos una variable la cual va almacenar el numero que el usuario va ingresar al programa cuando este en ejecución (hacemos un casteo del tipo de dato **int** ya que Python los datos ingresados son **str**).

Ya por último mandamos imprimir el numero y mandamos a llamar la función y pasamos por parámetro la variable **número**.

Diagrama y resultado



Función 2

Función que calcule el valor factorial de un numero entero positivo.

Código

```python
def factorial(num):
    numFactorial = 1

    while(num > 0):
        numFactorial *= num
        num -= 1

    return numFactorial
numero = int(input("Numero: "))
print("Numero:",numero, "Numero factorial: ",factorial(numero),"!")
```

Explicación

Primero definimos lo que es la función **factorial()** la cual lleva como argumento **num**, con **num** será el número que quiérenos sacar el factorial. Definimos una variable **numFactorial** la cual inicializamos en 1, esto lo que va hacer es que será nuestra bandera la cual contendrá nuestros valores multiplicados ente si de num.

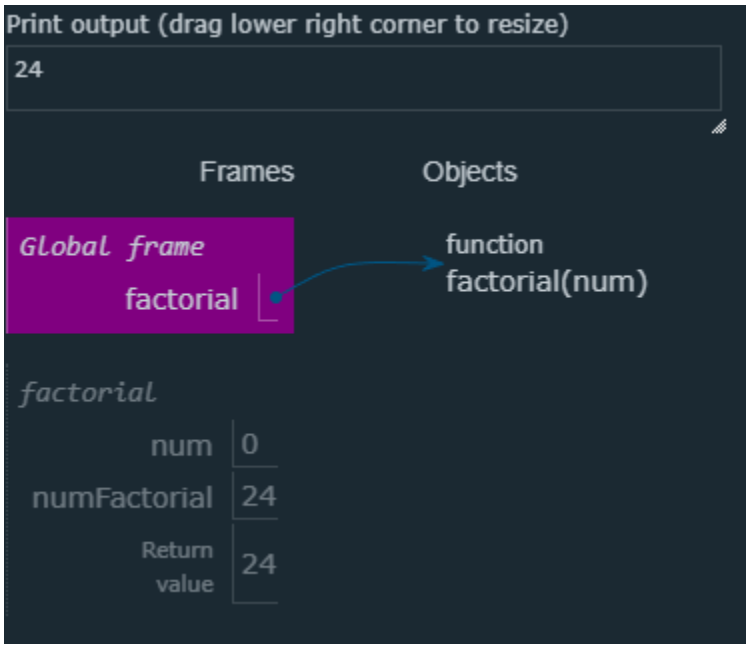
Creamos un **while** con una condición que es **num > 0** esto va controlar nuestro ciclo, tomamos nuestra variable bandera y la multiplicamos por el valor del parámetro **num**, después decrementamos en uno **num**.

Ya para terminar la definición de la función, solo retornamos el valor de nuestro numero ya factorial, así que retornamos nuestra variable bandera.

Declare una variable **numero** la cual contendrá el numero que el usuario quiere que sea factorial, le aplicamos un cateo de **int.**

Solo queda imprimir el numero que queremos hacer factorial, y el valor ya en factorial.

Diagrama y resultado



Función 3

Función que cuente cuantas veces se repite el patrón en una cadena de caracteres incluyendo si se superponen.

Código

```python
import re

def cuenta_patron(buscar,texto):
    listTexto = re.findall(r"(?=("+buscar+"))",texto)
    return len(listTexto)

buscar = input("Texto a buscar: ")
texto = input("Texto: ")

print("Numero de repeticiones:",cuenta_patron("aba","gabababa"))

```

Explicación

Primero importamos **re** es una librería de Python que nos es posible trabajar con expresiones regulares.

Después creamos la función **cuenta_patron()** la cual contendrá dos argumentos los cuales será el texto a buscar y el texto donde verificaremos cuantas veces se repite lo que buscamos.

Dentro de la función declaramos una variable done almacenara una lista de toda de elementos los cuales se repiten (también cuentan los que se sobreponen).

¿La "magia" de esto es la expresión regular la que utilizamos la cual es una búsqueda posterior positiva lo cual nos está definida como **?=** lo cual la funcionalidad de esta es lo que va inmediatamente después de la posición actual de la cadena es el **texto a buscar** que es un grupo a buscar **(aba).**

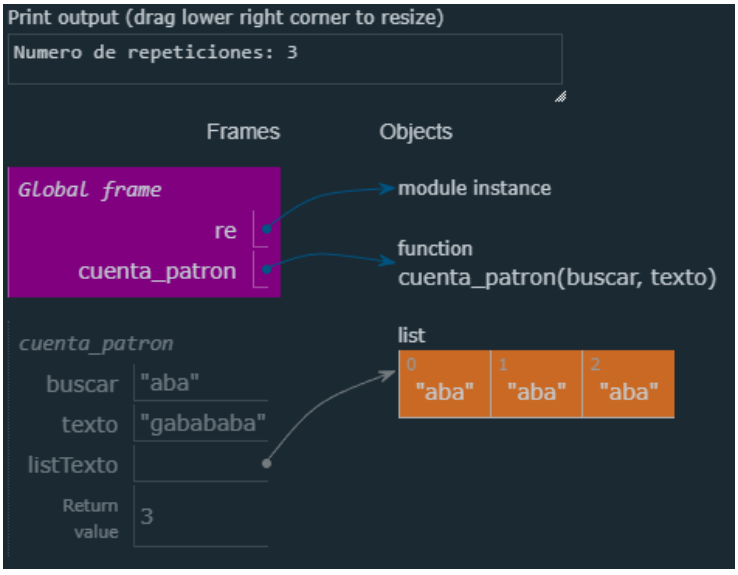
La función **.findall()** lo que va hacer es buscar en toda la cadena pasada por el parámetro **buscar.**

Solo quedaría retornar cuantos grupos se encontraron en la cadena, para ello tenemos que usar un **len()** en la variable **listTexto** la cual contiene nuestra lista de los grupos buscados compatibles establecidos por nuestros parámetros.

Solo queda pedir los datos al usuario y para ello declaramos dos variables las cuales son **buscar** y **texto** los cuales le colocamos un **input()** para el ingreso de los caracteres.

Solo quedaría mandar llamar la función y pasar esas variables como parámetros y listo.

Diagrama y resultado



Función 4

Función que devuelva la parte de un árbol de acuerdo con su índice.

Código

```python
arbol = [[[1,2],3],[4,[5,6]],[7],[8,9,10]]

def arbol_ref(arbol, tupla):

    listaArbol = list(tupla)
    n = len(listaArbol)

    if(n == 1):
        print(arbol[listaArbol[0]])
    if(n == 2):
        print(arbol[listaArbol[0]][listaArbol[1]])
    if(n == 3):
        print(arbol[listaArbol[0]][listaArbol[1]][listaArbol[2]])

arbol_ref(arbol,(1,))

```

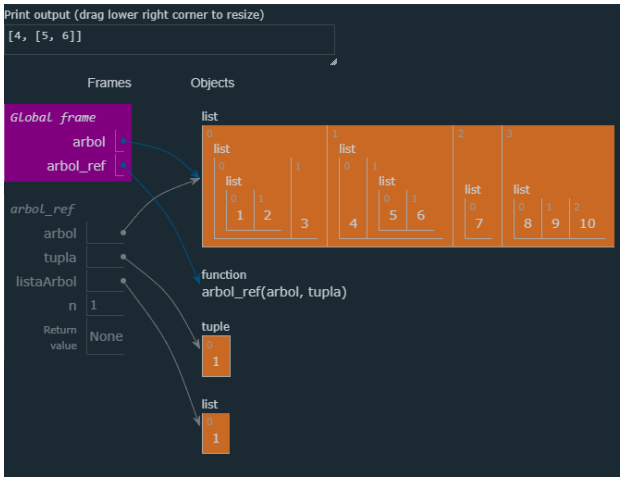
Explicación

Primero definimos nuestro árbol con el cual vamos a pasar como parámetro a nuestra función

La función lo que hace es obtener la tupla como argumento y convertirla en una lista para poder trabajar cómodamente, después de esto evaluaremos nuestros casos

Si n = 1 esto nos quiere decir que tenemos que mostrar toda esa rama

Si n = 2 tenemos que mandar a llamar un nodo en específico al igual que en el 3



Función 5

Función que desarrolle una expresión algebraica.

Código

```python
from sympy import expand

def expresion(e):
    return expand(e)

userExpresion = input("Ingrese la expresion: ")

print("La expresion: ",userExpresion,"Simplificada es: ", expresion(userExpresion))
```

Explicación

Para hacer esta función tenemos que hacer uso de la biblioteca **sympy** que es para la sibologia matemática, para ello tenemos que instalara previamente la cual el comando es **pip install sympy**

Ya instalada solo queda importar **expand** lo que hace esta es para expandir una expresión algebraica.

Solo queda pedir la expresión que será un string y será almacenada en una variable llamada **userExpresion** la cual será pasada como parámetro a la llamada a la función.

Diagrama y resultado