COLOMBIA POTENCIA DE LA VIDA

TIC

Módulo 1

LECCIÓN 4

Recursividad

Recursividad

Una función recursiva es aquella que está definida en función de sí misma, por lo que es una técnica donde una función se invoca a si misma repetidamente hasta llegar a un punto de salida.

```
Tanto el factorial como la serie Fibonacci,
son las mejores opciones para ilustrar este fenómeno.
```

Si no se considera una condición que detenga las invocaciones recursivas, el programa puede entrar en un bucle infinito.

Ejercitación Factorial y Fibonacci

Para comprender mejor la recursividad es importante que tenga en mente la función factorial y como se calculan sus números, por ejemplo a cuanto equivale un 5! y la serie Fibonacci.

Su tarea es investigar sobre estos dos elementos, encontrar le valor factorial solicitado y los primeros 8 números de la serie de Fibonacci.

```python
[ ]  def factorial_function(n):
        if n < 0:
            return None
        if n < 2:
            return 1

        product = 1
        for i in range(2, n + 1):
            product *= i
        return product

    for n in range(1, 6): # probando
        print(n, factorial_function(n))
```

```
1 1
2 2
3 6
4 24
5 120
```

```
[ ]  def fib(n):
         if n < 1:
             return None
         if n < 3:
             return 1
         elem_1 = elem_2 = 1
         the_sum = 0
         for i in range(3, n + 1):
             the_sum = elem_1 + elem_2
             elem_1, elem_2 = elem_2, the_sum
         return the_sum

     for n in range(1, 9): # probando
         print(n, "->", fib(n))
```

```
1 -> 1
2 -> 1
3 -> 2
4 -> 3
5 -> 5
6 -> 8
7 -> 13
8 -> 21
```

¿Coinciden con los cálculos realizados?

Hasta allí las funciones están construidas de manera tradicional, no utilizan recursividad.

Hagamos una función recursiva entonces, una función que sume los números desde 1 hasta 10, según Gauss, la sumatoria de los n primeros números se puede calcular con una formula como la siguiente k = n(n+1)/2, para el caso de 1 a 10, el resultado sería 55.

Pero como la tarea es crear una función recursiva(aunque para este caso no tenga mucho beneficio), hay que definir la condición de parada, por ejemplo definimos que el usuario indica el valor final de la sumatoria y vamos a decrementarlo en cada iteración hasta que ese valor sea igual a 1, sumando su valor en cada iteración, así entonces, la condición de parada será que n=1.

```
n es igual a 10
si n es igual a 1 entonces retorno 1
si no se cumple la condición anterior
retorne n sumado a llamar nuevamente a la función con n - 1
```

```
[ ]  #Versión recursiva
     def suma_n(n):
       if n == 1:
         return 1
       return n + suma_n(n - 1)

     print(suma_n(10))

     #Versión de Gauss
     def suma_gauss(n):
       return n*(n+1)/2

     print(suma_gauss(10))

     #Cómo sería la versión utilizando for y range?, esa es su tarea
     def suma_range(n):
       pass
```

```
55
55.0
```

Ejercitación recursividad

Ya conoce la función factorial y la sucesión de Fibonacci, incluso una implementación tradicional con funciones, su tarea es implementar dichas funciones utilizando recursividad.

Ejercitación El juego del ahorcado

Solución

Se viene la construcción de un proyecto que aplica o con el que se practicará todos los temas vistos, como adicional, se crearán dos módulo que se importarán, similar a como se ha trabajado con el módulo random.

Los conocimientos necesarios son:

- Ciclos For y While
- Condicionales if-else
- Listas
- Cadenas
- range
- El uso básico de

El juego del ahora es simple, el jugador debe adivinar una palabra, seleccionada aleatoriamente de una lista de palabras, el usuario verá espacios vacíos (guiones bajos) que representan la cantidad de caracteres de la palabra. Para adivinar la palabra el usuario debe indicar una letra a la vez, si la letra se incluye en la palabra, deberá reemplazar el espacio vacío por la letra correspondiente, tantas como contenga la palabra, en caso de la letra no encontrarse en la palabra, se deberá presentar un muñeco al que se le irán pintando sus extremidades hasta completarlo, una extremidad por cada error del usuario.

El juego termina cuando el jugador completa la palabra antes de ser ahorcado y se dará como ganador o cuando el muñeco del usuario sea dibujado completamente y se dará como perdedor.

El proyecto se desarrollará en 3 archivos, el archivo main que contiene toda la lógica del juego, el archivo de palabras, que contiene una lista con todas las palabras con las que se va a jugar y el archivo del arte, que contendra una lista con los dibujos del ahorcado en cada una de sus etapas.

Por ejemplo:

arte.py

```
# final ahorcado

  +---+
  |   |
  o   |
 /|\  |
 / \  |
      |

========
```

palabras.py

```
word_list = [
'abruptly',
'aeroplano'
]
```

main.py

```
import random
from palabras import word_list
from arte import etapas
.
.
```

Inicie construyendo un diagrama de flujo y recuerde el concepto divide y vencerás.

Manejo de Ficheros

El manejo de archivos es una parte muy importante de cualquier aplicación, ya sea para la escritura de logs como vbitácora de acciones o errores, o para el almacenamiento de datos.

En python existen un par de maneras de hacer, la primera y que tiene integrada entre sus funciones propias es el método open()

El método cuenta con las opciones que se presentan a continuación, aunque no todas son obligatorias.

```
open(file, mode='r', buffering=- 1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
```

- **file** es la ruta hacía el fichero que se quiere abrir, puede ser relativa o absoluta.
- **mode** define la forma en la cual se abrirá el archivo, por defecto se abre en modo 'r' que es solo lectura pero cuenta con más opciones.

Opción	Significado
'r'	abierto para lectura (por defecto)
'w'	abierto para escritura
'x'	abierto para creación en exclusiva, falla si el fichero ya existe
'a'	abierto para escritura, añadiendo al final del fichero si este existe
'b'	modo binario
't'	modo texto (por defecto)
'+'	abierto para actualizar (lectura y escritura)

- **buffering** es opcional y. permite configurar la política del buffer

- **encoding** es el nombre de la codificación empleada con el fichero. Esto solo debe ser usado en el modo texto, por ejemplo UTF-8

- **errors** es una cadena opcional que especifica como deben manejarse los errores de codificación y descodificación – esto no puede ser usado en modo binario. Están disponibles varios gestores de error. Cuenta con opciones como:
  - 'strict' para lanzar una excepción ValueError si hay un error de codificación. El valor por defecto, None, produce el mismo efecto.
  - 'ignore' ignora los errores. Nótese que ignorar errores de codificación puede conllevar la pérdida de datos.
  - 'replace' provoca que se inserte un marcador de reemplazo (como '?') en aquellos sitios donde hay datos malformados.
  - 'surrogateescape' representará cualquier bytes incorrectos como unidades de código sustituto bajo que van desde U+DC80 a U+DCFF. Estas unidades de código sustituto volverán a convertirse en los mismos bytes cuando el gestor de errores surrogateescape sea usado al escribir datos. Esto es útil para el procesado de ficheros con una codificación desconocida.
  - 'xmlcharrefreplace' está soportado solamente cuando se escribe a un fichero. Los caracteres que no estén soportados por la codificación son reemplazados por la referencia al carácter XML apropiado &#nnn;.
  - 'backslashreplace' reemplaza datos malformados con las secuencias de escapes de barra invertida de Python.
  - 'namereplace' reemplaza caracteres no soportados con secuencias de escape \N{...} (y también está sólo soportado en escritura).
- **newline** determina cómo analizar los caracteres de nueva línea de la secuencia. Puede ser None, '', '\n', '\r', y '\r\n'.
-

Los archivos, como buena practica, es importante cerrarlos después de operar, para ello se utiliza el comando close()

```
[ ]  f = open('/content/sample_data/anscombe.json', 'rt')

     print(f.read(40)) #Puede leer todo el archivo con solo read() o indicar un número de carácteres

     #o readline para leer una línea completa
     print(f.readline())
     print(f.readline())
     print(f.readline())
```

```
     # O ciclar por todo el archivo
     for line in f:
       print(line)
```

```
     [
       {"Series":"I", "X":10.0, "Y":8.04},

       {"Series":"I", "X":8.0, "Y":6.95},

       {"Series":"I", "X":13.0, "Y":7.58},

       {"Series":"I", "X":9.0, "Y":8.81},
```

```
[ ]  # Para escribir en un archivo, según la tabla anterior utilizamos la opción
     #'a' para añadir al final o 'w' para sobrescribir el archivo

     f = open("demofile2.txt", "a") #Crear el archivo si no existe en la ruta indicada
     f.write('Mis primeras líneas en el fichero \n')
     f.close()

     f = open("demofile2.txt", "r")
     print(f.read())
```

```
     Mis primeras líneas en el ficheroMis primeras líneas en el ficheroMis primeras líneas en el fichero
     Mis primeras líneas en el fichero \nMis primeras líneas en el fichero
```

```
[ ]  f = open("demofile2.txt", "w")
     f.write("Uy, qué pasó?\n")
     f.close()

     #Ahora se ha sobrescrito el contenido anterior:
     f = open("demofile2.txt", "r")
     print(f.read())
```

```
     Uy, qué pasó?
```

```
[ ]  # con la opción 'x' se crea el archivo solo si no existe, caso contrario da un error
     f = open("demofile2.txt", "x")
     f.write("Como ya existe, esto no va a funcionar\n")
     f.close()

     #Ahora se ha sobrescrito el contenido anterior:
     f = open("demofile2.txt", "r")
     print(f.read())
```

Ya se han creado, leído y actualizado archivos, lo que resta es la eliminación de ficheros, para hacerlo es necesario utilizar el módulo os y su método remove().

Para eliminar un archivo lo primero es verificar que el archivo exista, así se evita un error en tiempo de ejecución.

Lo mismo aplica para un directorio completo, solo que ahora se utiliza el método rmdir(), teniendo en cuenta que la carpeta debe estar vacía.

```
[ ]  import os
     #f = open("demofile23.txt", "x")

     if os.path.exists("demofile23.txt"):
       print("The file was deleted")
       os.remove("demofile23.txt")
     else:
       print("The file does not exist")
```

```
[ ]  dir(f)
     help(os)
```