Trabajo práctico nro. 8

UTN FACULTAD REGIONAL MENDOZA	**Asignatura: Programación I**	
	Cursado: Primer Trimestre	**Horas semanales**: 4
	Carrera: *Tecnicatura Universitaria en Programación*	**Nivel (Año):** ☐ 1° ☐ 2° ☐ 3°
	Ciclo Lectivo: 2023	

Integrantes de la Cátedra:

- **DOCENTES:**

Nombre del Profesor	Periodo	Cantidad horas materia
		6 horas

1. Escribir una función que reciba un número positivo n y devuelva la cantidad de dígitos que tiene.

```python
def count_digits(n):
    if n < 10:
        return 1  # Caso base: cuando n tiene un solo dígito
    else:
        return 1 + count_digits(n // 10)  # Llamada recursiva

# Solicitar al usuario un número positivo
number = int(input("Ingresa un número positivo: "))

# Llamar a la función contar_digitos y mostrar el resultado
quantity_digits = count_digits(number)
print(f"El número {n} tiene {quantity_digits} dígitos.")

```

2. Escribir una función que reciba 2 enteros n y b y devuelva True si n es potencia de b.

```python
def is_power(n, b):
    if n == 1:
        return True  # Caso base: 1 es siempre una potencia de cualquier número
    if n < b:
        return False  # Cuando n es menor que b y no es igual a 1, no es una potencia de b
    if n % b != 0:
        return False  # Si n no es divisible exactamente por b, no es una potencia de b
    return is_power(n // b, b)  # Llamada recursiva

# Solicitar al usuario dos enteros, n y b
n = int(input("Ingresa un número entero n: "))
b = int(input("Ingresa un número entero b: "))

# Llamar a la función es_potencia y mostrar el resultado
resultado = is_power(n, b)
if resultado:
    print(f"{n} es una potencia de {b}.")
else:
    print(f"{n} no es una potencia de {b}.")
```

3. Escribir una funcion recursiva que reciba como parámetros dos strings a y b, y devuelva una lista con las posiciones en donde se encuentra b dentro de a. Ejemplo:

```
>>> posiciones_de("Un tete a tete con Tete", "te")
[3, 5, 10, 12, 21]
```

```python
def positions_of(a, b, start=0, positions=None):
    if positions is None:
        positions = []

    # Encontrar la próxima ocurrencia de b en a a partir de la posición 'inicio'
    index = a.find(b, start)

    if index != -1:
        positions.append(index)
        return positions_of(a, b, index + 1, positions)  # Llamada recursiva para buscar más ocurrencias
    else:
        return positions

# Ejemplo de uso
string_a = "Un tete a tete con Tete,te"
string_b = "te"

result = positions_of(string_a, string_b)
print(result)  # Deberia imprimir [3, 5, 10, 12, 21]
```

4. Escribir dos funciones mutualmente recursivas par(n) e impar(n) que determinen la paridad del numero natural dado, conociendo solo que:
    - 1 es impar.
    - Si un número es impar, su antecesor es par; y viceversa.

```python
def par(n):
    if n == 0:
        return True  # 0 se considera par
    else:
        return impar(n - 1)

def impar(n):
    if n == 0:
        return False  # 1 se considera impar
    else:
        return par(n - 1)

# Ejemplos de uso
number = int(input("Ingrese un número: "))

if par(number):
    print(f"{number} es par.")
else:
    print(f"{number} es impar.")
```

5. Escribir una función recursiva que encuentre el mayor elemento de una lista.

```python
def find_major(lista):
    if len(lista) == 1:
        return lista[0]  # Caso base: cuando la lista tiene un solo elemento, ese es el mayor
    else:
        # Encontrar el mayor entre el primer elemento y el mayor del resto de la lista
        first_element = lista[0]
        major_rest = find_major(lista[1:])
        if first_element > major_rest:
            return first_element
        else:
            return major_rest

# Ejemplo de uso
mi_lista = [3, 7, 1, 12, 5, 9]

major = find_major(mi_lista)
print(f"El mayor elemento de la lista es: {major}")
```

6. Escribir una función recursiva para replicar los elementos de una lista una cantidad n de veces. Por ejemplo, replicar ([1, 3, 3, 7], 2) = ([1, 1, 3, 3, 3, 3, 7, 7])

```python
def reply(lista, n):
    if n == 1:
        return lista  # Caso base: replicar una vez es la lista original
    else:
        return lista + reply(lista, n - 1)  # Llamada recursiva

# Ejemplo de uso
my_list = [1, 3, 3, 7]
n = 2

replied_list = reply(my_list, n)
print("Lista replicada:", replied_list)
```

7. Implemente un algoritmo, usando una función recursiva, que resuelva la siguientesumatoria: K(n, p) = p + 2 ∗ p + 3 ∗ p + 4 ∗ p + … + n ∗ p
   - El programa debe pedir al usuario que ingrese un número n, y un número d,
   - Luego debe calcular el valor de K(n, p) usando una función recursiva,
   - Debe imprimir el resultado de K(n, p)

```python
def calculate_sum(n, p):
    if n == 1:
        return p  # Caso base: cuando n es 1, el resultado es p
    else:
        return n * p + calculate_sum(n - 1, p)  # Llamada recursiva

# Solicitar al usuario que ingrese los valores de n y p
n = int(input("Ingresa un número n: "))
p = int(input("Ingresa un número p: "))

# Calcular la sumatoria K(n, p) utilizando la función recursiva
result = calculate_sum(n, p)

# Mostrar el resultado de K(n, p)
print(f"El resultado de K({n}, {p}) es: {result}")
```

8. El triángulo de Pascal es un arreglo triangular de números que se define de la siguiente manera: Las filas se enumeran desde n = 0, de arriba hacia abajo. Los valores de cada fila se enumeran desde k = 0 (de izquierda a derecha). Los valores que se encuentran en los bordes del triángulo son todos unos. Cualquier otro valor se calcula sumando los dos valores contiguos de la fila de arriba. Escribí la función recursiva pascal(n, k) que calcula el valor que se encuentra en la fila n y la columna k.

```python
def pascal(n, k):
    if k == 0 or k == n:
        return 1  # Los valores en los bordes del triángulo son siempre 1
    else:
        return pascal(n - 1, k - 1) + pascal(n - 1, k)  # Llamadas recursivas

# Solicitar al usuario que ingrese los valores de n y k
n = int(input("Ingresa el valor de n: "))
k = int(input("Ingresa el valor de k: "))

# Calcular el valor en el Triángulo de Pascal utilizando la función recursiva
result = pascal(n, k)

# Mostrar el resultado
print(f"El valor en la fila {n} y la columna {k} del Triángulo de Pascal es: {result}")
```

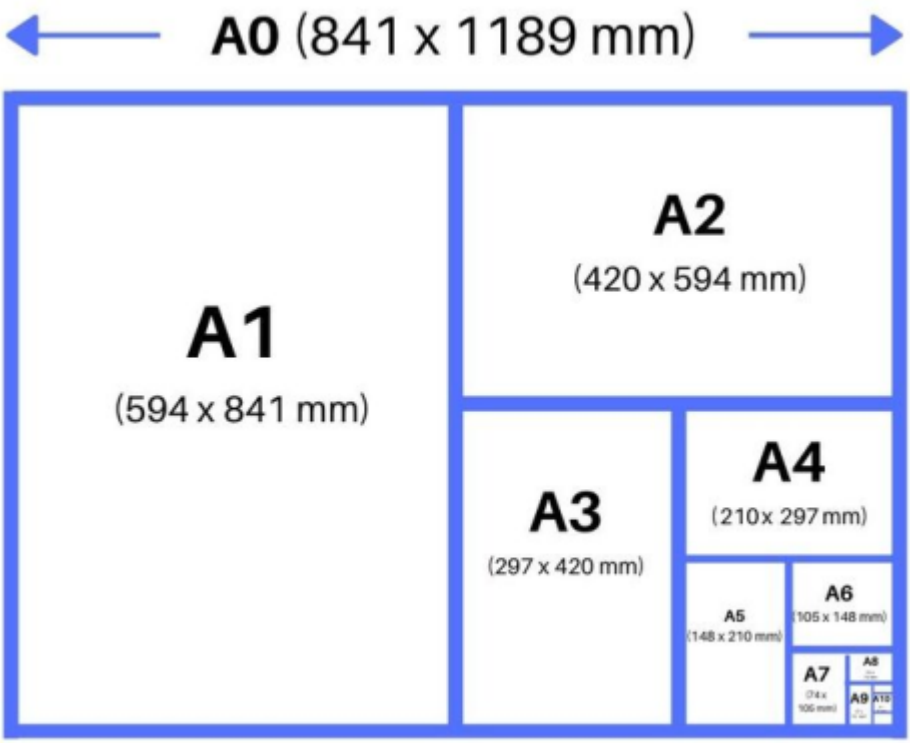
9. Escribí una función recursiva combinaciones(lista, k) que reciba una lista de caracteres únicos, y un número k, e imprima todas las posibles cadenas de longitud k formadas con los caracteres dados (permitiendo caracteres repetidos).

```python
def combinations(lista, k, actual_string=""):
    if k == 0:
        print(actual_string)
    else:
        for caracter in lista:
            combinations(lista, k - 1, actual_string + caracter)

# Ejemplo de uso
list_of_caracters = ['A', 'B', 'C']
lenght = 3

combinations(list_of_caracters, lenght)
```

10. La norma ISO 216 especifica tamaños de papel. Es el estándar que define el popular tamaño de papel A4 (210 mm de ancho y 297 mm de largo). Las hojas A0 miden 841 mm de ancho y 1189 mm de largo. A partir de la A0 las siguientes medidas, digamos la A(N+1), se definen doblando al medio la hoja A(N). Siempre se miden en milímetros con números enteros: entonces la hoja A1 mide 594 mm de ancho (y no 594.5) por 841 mm de largo.



Escribí una función recursiva medidas_hoja_A(N) que para una entrada N mayor que cero, devuelva el ancho y el largo de la hoja A(N) calculada recursivamente a partir de las medidas de la hoja A(N−1), usando la hoja A0 como caso base. La función debe devolver el ancho y el largo -en ese orden- en una tupla.

C: > Users > Tity Lacoste > OneDrive > Escritorio > x=46.py > ...

```python
def papers_size(N):
    if N == 0:
        return (841, 1189)  # Caso base: Hoja A0
    else:
        width_A_N_1, hight_A_N_1 = papers_size(N - 1)
        width_A_N = hight_A_N_1
        hight_A_N = width_A_N_1 * 2
        return (width_A_N, hight_A_N)

# Tamaño que desea calcular
N = int(input("Ingrese el tamaño que desea calcular: "))

width, hight = papers_size(N)
print(f"Ancho de la hoja A{N}: {width} mm")
print(f"Largo de la hoja A{N}: {hight} mm")

```