Actividad Guiada 1

Autor: Roger Amorós Sirera

https://github.com/RogerAmoros13/algoritmos_de_optimizacion

En el repositiorio esta subido el fichero *hanoi_app.py* donde esta implementado el algoritmo de Hanoi de manera visual que se resuelve solo de manera no recursiva.

In [1]:
```python
# Ejercicio 1

# Algoritmo recursivo para resolver la Torre de Hanoi visto en clase
def hanoi_tower(n, orig, piv, dest):
    if n == 1: # Condición de retorno de las llamadas recursivas
        print(f"Mover disco de {orig} a {dest}")
        return
    hanoi_tower(n-1, orig, dest, piv)
    print(f"Mover disco de {orig} a {dest}")
    hanoi_tower(n-1, piv, orig, dest)
```

In [2]:
```python
hanoi_tower(3, 1, 2, 3)
```

```
Mover disco de 1 a 3
Mover disco de 1 a 2
Mover disco de 3 a 2
Mover disco de 1 a 3
Mover disco de 2 a 1
Mover disco de 2 a 3
Mover disco de 1 a 3
```

In [3]:
```python
# Ejercicio 2

# Función para calcular el cambio con la menor cantidad de monedas.
# Ventajas sobre el algoritmo visto en clase:
#   - No se precalcula el array de soluciones.
#   - Se itera directamente sobre las monedas disponibles --> O(n) con n = len(monedas_disp)
# Se modifica el valor pasado, pero se reduce la cantidad de variables almacenadas en memoria
def cambio_monedas(valor, monedas_disp):
    result = []
    for moneda in monedas_disp:
        times_used = valor // moneda
        valor -= moneda * times_used
        result.append(times_used)
    return result

print(cambio_monedas(27, [10, 7, 3, 2])) # Ejemplo random
print(cambio_monedas(177, [50, 20, 10, 5, 2, 1])) # Monedas enteras convencionales
print(cambio_monedas(177.65, [50, 20, 10, 5, 2, 1, .5, .2, .1, .05])) # Funciona con decimales
```

```
[2, 1, 0, 0]
[3, 1, 0, 1, 1, 0]
[3.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0]
```

In [4]:
```python
# Ejercicio 2

import numpy as np

# Tenemos que comparar todos los puntos entre si, por lo que es practicamente
# obligatorio realizar dos bucles.
# No obstante, podemos evitar varias comprobaciones
# - Un punto consigo mismo.
# - Un punto con otro pero en orden distinto.

# Para esto, iteraremos primero por los elementos de la lista hasta el penultimo,
# y anidaremos otro bucle que vaya desde el siguiente que se esta evaluando en el
# bucle superior al último punto.

# Vemos las comprobaciones realizadas con una lista de 4 puntos [1, 2, 3, 4]:
# - 1. (1, 2)
# - 2. (1, 3)
# - 3. (1, 4)
# - 4. (2, 3)
# - 5. (2, 4)
# - 6. (3, 4)

def distancia_euclidea(point1, point2):
    if isinstance(point1, int):
        point1 = [point1]
    if isinstance(point2, int):
        point2 = [point2]
    _sum = 0
    for p1, p2 in zip(point1, point2):
        _sum += (p1 - p2)**2
    return np.sqrt(_sum)

def closest_points(point_list):
    closest_point = 1e20 # Suficientemente grande
    best_sol = []
    len_list = len(point_list)
    for i in range(len_list - 1):
        for j in range(i + 1, len_list):
            dist = distancia_euclidea(
                point_list[i], point_list[j]
            )
            if dist < closest_point:
                best_sol= [point_list[i], point_list[j], dist]
                closest_point = dist
    return best_sol
```

```
print(closest_points([(1, 12), (4, 5), (20, 40), (13, 40), (35, 12)]))
```

```
[(20, 40), (13, 40), 7.0]
```

In [ ]: