Condicionales. Ejercicios y soluciones

Ejercicio 1. La dura tarea de poner las notas

Diseña una función que dé la calificación (“suspenso”, …, Matrícula de Honor”) correspondiente a una nota, exigiendo que la entrada sea un número real entre 0 y 10.

In [1]:

```
def calificacion(nota):
    """
    Param:
        nota: float
    Pre:
        0 <= nota <= 10
    """
    if nota < 5.0:
        return "Suspenso"
    elif nota < 7.0:
        return "Aprobado"
    elif nota < 9.0:
        return "Notable"
    elif nota < 10.0:
        return "Sobresaliente"
    else:
        return "Matrícula de Honor"

for nota in [0.5, 4.999, 5.0, 5.5, 6.8, 7.2, 9.25, 10.0]:
    print(nota, calificacion(nota))
```

```
0.5 Suspenso
4.999 Suspenso
5.0 Aprobado
5.5 Aprobado
6.8 Aprobado
7.2 Notable
9.25 Sobresaliente
10.0 Matrícula de Honor
```

Ecuación de segundo grado

Diseña un programa que calcule las soluciones de una ecuación de la forma $ax^2 + bx + c = 0$. Debes tener en cuenta que los coeficientes pueden ser reales arbitrarios, nulos o no, etc., dando lugar a una ecuación sin solución, con una o con dos soluciones reales o imaginarias.

Añadimos por nuestra cuenta

In [1]:

```
import math, cmath

def resuelve_ec_2_grado(a, b, c):
    """
    ...
    Parameters:
    -----------
        a, b, c: float, float, float
    Returns:
    --------
        (String, [Float])
        Number and type of solutions, [solutions]
    ...
    """
    if a == 0:
        if b == 0:
            if c == 0:
                return ("Inf sols",[])
            else:
                return ("Sin sol",[])
        else:
            return ("Sol única", [-c/b])
    else:
        discriminante = b**2 - 4*a*c
        if discriminante >= 0:
            sol1 = (-b + math.sqrt(discriminante)) / (2*a)
            sol2 = (-b - math.sqrt(discriminante)) / (2*a)
            return ("2 solucs reales", [sol1, sol2])
        else:
            sol1 = (-b + cmath.sqrt(discriminante)) / (2*a)
            sol2 = (-b - cmath.sqrt(discriminante)) / (2*a)
            return ("2 solucs imags", [sol1, sol2])

print("a ", resuelve_ec_2_grado(0, 0, 0))
print("b ", resuelve_ec_2_grado(0, 0, 3))
print("c ", resuelve_ec_2_grado(0, 2, 5))
print("d ", resuelve_ec_2_grado(1, -5, 6))
print("e ", resuelve_ec_2_grado(1, 2, 5))
```

```
a  ('Inf sols', [])
b  ('Sin sol', [])
c  ('Sol única', [-2.5])
d  ('2 solucs reales', [3.0, 2.0])
e  ('2 solucs imags', [(-1+2j), (-1-2j)])
```

Códigos de rotación

Imaginemos las letras del alfabeto ordenadas y dispuestas en círculo. Esto es, a la derecha de la A se encuentra la B, luego la C y así sucesivamente, hasta la Z, y a la derecha de la Z se encuentra nuevamente la A. Definimos una función que codifica un carácter c según un desplazamiento k, moviendo ese carácter c, k posiciones a la derecha.

Defínela y úsala para codificar una frase.

In [2]:

```
def cod_char(a, k):
    if not a.isalpha():
        return a
    kk = ord(a) + k%26
    aa = chr(kk)
    if a.isupper() and aa.isupper() or a.islower() and aa.islower():
        return aa
    else:
        return chr(kk - 26)

def cod_linea(linea, k):
    return "".join([cod_char(a, k) for a in linea ])

frase = "Al ataque."
codificada = cod_linea(frase, 7)
descodificada = cod_linea(codificada, -7)

print(frase)
print(codificada)
print(descodificada)

codificada = cod_linea(frase, 2000)
descodificada = cod_linea(codificada, -2000)
print(codificada)
print(descodificada)
```

```
Al ataque.
Hs hahxbl.
Al ataque.
Yj yryosc.
Al ataque.
```

In [3]:

```
# Otra versión:

import string
def cod_char(a, k):
    if not a.isalpha():
        return a
    if a.isupper():
        alfabeto = string.ascii_uppercase
    else:
        alfabeto = string.ascii_lowercase
    kk = (ord(a) - ord(alfabeto[0]) + k)%26
    return alfabeto[kk]

def cod_linea(linea, k):
    return "".join([cod_char(a, k) for a in linea ])

frase = "Al ataque, el enemigo va a ser vencido."
codificada = cod_linea(frase, 7)
descodificada = cod_linea(codificada, -7)

print(frase)
print(codificada)
print(descodificada)

codificada = cod_linea(frase, 2000)
descodificada = cod_linea(codificada, -2000)
print(codificada)
print(descodificada)
```

```
Al ataque, el enemigo va a ser vencido.
Hs hahxbl, ls lultpnv ch h zly clujpkv.
Al ataque, el enemigo va a ser vencido.
Yj yryosc, cj clckgem ty y qcp tclagbm.
Al ataque, el enemigo va a ser vencido.
```