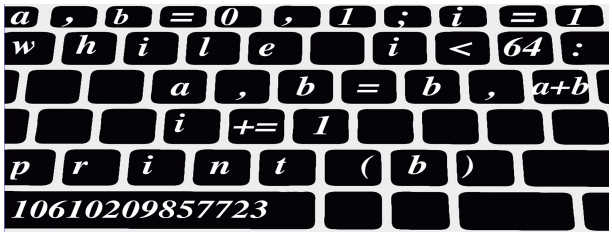

FUNDAMENTOS DE COMPUTACIÓN
Grado en ingeniería en tecnologías Industriales
Universidad de Cantabria

Examen Final(parte escrita)
14 de Enero del 2021

Nombre y apellidos:

---

1. [12 %] Contesta brevemente a las siguientes cuestions

   1. Si el reloj de la CPU de un computador es de 2MHz, y se requiere ejecutar un programa de Python de 10 instrucciones, ¿cuánto tiempo tardará en ejecutarse?

   2. ¿Cúal es la diferencia entre un `interprete` y un `compilador`? Escribe dos lenguajes interpretados y otros dos compilados.

   3. ¿Cúal es la diferencia entre los operadores de Python: `is` y `==`?

2. [12 %] ¿Qué se muestra por pantalla después de ejecutar estos dos programas ?

```
##PROGRAMA 1

x = 2
try:
   print('Parte A')
   assert x < 0, 'Fallo'
   print('Parte B')
except:
   print('Parte C')
   print('Parte D')

SOLUCION(PROGRAMA 1):
```

```
##PROGRAMA 2

i = 1
to = "altocampoo"
while i < len(to):
    to += to[len(to)-i]
    i *= i+1
print(to)

SOLUCION(PROGRAMA 2):
```

3. [14 %] Completar los puntos suspensivos del programa `prog2.py`. No se permite alterar la estructura del bucle, y atendiendo a las siguientes especificaciones.

   1. La primera función `busqueda_binaria1` tiene como argumentos: un dato númerico `a` y, una lista ordenada de números `L`. Devuelve la **primera** posición del elemento `a` en `L`. En caso contrario devuelve -1.
   2. La segunda función `busqueda_binaria2` tiene cuatro argumentos: un número `a`, una lista de números `L`; y números enteros h, k, verificando que la lista `L[h:k]` está ordenada. Devuelve la **última** posición del elemento `a` en la lista `L`, si `a` está en `L[h:k]`. En caso contrario devuelve -1.

prog1.py

```
def busqueda_binaria1(a, L):
    i = 0
    j = len(L)
    mitad = (i+j)//2
    while i < j:
        if L[mitad] < a:
            i = mitad +1
        else:
            j = mitad
        mitad = (i +j)//2
    encontrado = (i < len(L) and L[i] == a)
    if encontrado:
        return i
    else:
        return -1
```

prog2.py

```
def busqueda_binaria2(a, L, h, k):
    ...
    ...
    mitad = ...
    while ...
        ...
        ...
        ...
        ...
        ...
    encontrado = ...
    if encontrado:
        return i
    else:
        return -1
```

```
>>> L=[1,2,2,2,5]
>>> busqueda_binaria1(2, L)
1
>>> busqueda_binaria2(2, L, -1,4)
3
```

```
>>> busqueda_binaria2(2,L,4,5)
-1
>>> G=[2,6,2,2,5]
>>> busqueda_binaria2(2,G,2,5)
3
```

4. [12 %] Se considera el siguiente programa:

```
x = [ ('1', 100, 20, 10), ('2', 150, 40, 20), ('3', 200, 50, 10) ]
y = { x[i][0]:x[i][1:3] for i in range(len(x)) }
print('y ',y)
z = {}
for i in range(len(x)):
     z[x[i][0]] = x[i][1:3]
     print('z',z)
```

- ¿Qué clase de objeto almacena las variables x, y, z ?

- ¿ Qué se muestra por pantalla después de ejecutar el programa?

5. [10 %] Se considera la clase **Color** que tiene tres atributos `rojo`, `verde` y `azul`. Donde cada uno de ellos es un entero entre 0 y 255.

```
class Color(object):
    def __init__(self, rojo=0, verde =0, azul = 0):
        self.rojo = rojo
        self.verde = verde
        self.azul = azul
```

Se pide escribir el método `clarear` que *aclara* cada atributo en un 10 %

```
def clarear(sef):
```

```
>>> violeta=Color(230,9,25)
>>> violeta.clarear()
>>> violeta.rojo,violeta.verde,violeta.azul
(253, 10, 28)
>>> violeta.clarear()
>>> violeta.rojo,violeta.verde,violeta.azul
(255, 11, 31)
```

6. [40 %] El proceso denominado `invertir/sumar` de un número natural, consiste en invertir sus cifras y sumarlos. Por ejemplo, si $n = 2021$ devuelve $3223 = 2021 + 1202$. Y, si $n = 129$ el proceso devuelve el número $1050 = 129 + 921$.

   Podemos repetir este mismo proceso con los números resultantes:

   - 129 con dos iteraciones se convierte en capicúa: 129+921=1050, 1050 +0501=1551
   - 59 con tres iteraciones: 59+95 = 154, 154+451 = 605, 605+506 = 1111
   - 97 con seis iteraciones: 97+79=176, 176+671=847, 847+748=1595, 1595+5951=7546, 7546+6457=44044
   - 89 con 24 iteraciones se convierte en 8813200023188, también capicúa.

   El objetivo de este ejercicio, es escribir programas en PYTHON, encontrando el mínimo número de iteraciones necesarias de este proceso y devolviendo el número capicúa resultante.

   - [5 %] Escribir la función `invertir` que tenga como entrada un número natural y devuelva el número invertido.

     ```
     def invertir(n):
     ```

     ```
     #EJEMPLOS

     >>> invertir(234)
     432
     >>> invertir(1200)
     21
     ```

- [5 %]Escribir la función `capicua` que tenga como entrada un número natural y devuelva el booleano `True`, si el número es capicúa y `False` en caso contrario.

```
def capicua(n):
```

```
#EJEMPLOS

>>> capicua(2021)
False
>>> capicua(323)
True
```

- [15 %]Escribir la función iterativa `lychrel_iterativa` que tenga como argumento un número natural n y devuelva el número de iteraciones y, el número capicúa resultante. (Ayuda: Usar las dos funciones anteriores)

```
def lychrel_iterativa(n):
```

```
#EJEMPLOS
>>> lychrel_iterativa(2021)
(1, 3223)
>>> lychrel_iterativa(129)
(2,1551)
>>> lychrel_iterativa(97)
(6,44044)
```

- [15 %]Escribir la función recursiva `lychrel_recursiva` que tenga como argumento un número natural n y devuelva el número de iteraciones y, el número capicúa resultante. (Ayuda: Usar las dos primeras funciones)

```
def lychrel_recursiva(n):
```

```
#EJEMPLOS

>>> lychrel_recursiva(2021)
(1, 3223)
>>> lychrel_recursiva(129)
(2,1551)
>>> lychrel_recursiva(97)
(6,44044)
```

Si alguno de los números no se convierte en capicúa, repitiendo este proceso, entonces se dice que el número es de **Lychrel**. No se conoce ningún número de Lychrel, el primer número donde no se ha podido computar la iteración es el 196, https://www.p196.org/. El nombre Lychrel fue acuñado por Wade VanLandingham en 2001 viviendo con su novia, Cheryl, en Florida(EEUU), y es un anagrama aproximado al nombre Cheryl. Según se dice siguen viviendo juntos en Florida. **¿Una historia de amor?**