FUNDAMENTOS DE COMPUTACIÓN

Grado en ingeniería en tecnologías industriales

Universidad de Cantabria

Examen Final (Aula)

30 de enero del 2019

Nombre y apellidos:

---

1. Después de ejecutar el programa siguiente, ¿qué se muestra por pantalla? *(1 punto)*

```
class Pendiente(object):

    def __init__(self,n="Pendiente"):
        self.nombre = n

    def __str__(self):
        return self.nombre

class Poligono(Pendiente):

    def __init__(self,ss,n="Poligono"):
        Pendiente.__init__(self,n)
        self.lados = [float(s) for s in ss]

    def perimetro(self):
        s = 0
        for i in self.lados:
            s += i
        return s

class Triangulo(Poligono):

    def __init__(self,s1=1,s2=1,s3=1,n="Triangulo"):
        Poligono.__init__(self, [s1,s2,s3], n)

t = Triangulo(3,4,5)
p = Poligono([2,3,2,4])
print(t)  ------------------------->

print(t.perimetro()) ---------------->

print(p)----------------------------->

print(p.perimetro()) --------------->
```

2. ¿Qué se muestra por pantalla tras ejecutar el siguiente programa ? *(1 punto)*

```
B, C, D = 0, 0, 0
for A in [ 0, 1, 3, 5, 7, 9 ]:
      if A :
          if A%2 == A:
              B += 1
          else:
              C += 1
          if A%3 == 0:
              D += 1
              break
print(A)  -------------------->

print(B)  -------------------->

print(C)  -------------------->

print(D)  -------------------->
```

3. Considera el siguiente código. *(1 punto)*

```
roma_numeros = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
for k in roma_numeros:
        if roma_numeros[k] % 2 == 0:
          roma_numeros[k] *= 2
        else:
          roma_numeros[k] += 2
```

Escribe el valor de cada expresión, una vez ejecutado. Si se desencadena un error, escribe 'Error' :

```
print(len(roma_numeros))  ---------------->

print(roma_numeros['L'])    -------------->

print(roma_numeros['M'])  ---------------->

print(100 in roma_numeros) ------------>
```

4. Para cada una de las siguientes cuestiones, redondea la respuesta correcta. *(0.75 puntos)*

   1. ¿Aproximadamente cuántos bytes tiene un Terabyte?

      A) Un trillion bytes
      B) Mil Megabytes
      C) Mil Gigabytes
      D) Un millon Gigabytes
      E) Ninguna de las de arriba.

   2. ¿Quién inventó Python?

      A) Guido Python
      B) Monty Python
      C) Guido van Rossum
      D) Alan Turing
      E) Ninguna de las anteriores.

   3. ¿Cuántos bytes se necesitan para codificar un carácter con la codificación UTF-8?

      A) 1
      B) 8
      C) 128
      D) 256
      E) Ninguna de los valores de arriba.

5. Escribe un programa para resolver una clásica adivinanza: 'En una granja hay A cabezas y B patas entre gallinas y conejos, ¿cuántas gallinas y cuántos conejos hay en la granja?' *(1 punto)*

```
## AYUDA: Usar un bucle para iterar todas las posibles soluciones.
## NOTA: La adivinanza  no siempre tiene solución para todos los pares A y B.
A = int(input('¿Cuántas cabezas?  '))
B = int(input('¿Cuántas patas? '))
```

6. Se considera el siguiente código: *(1.5 puntos)*

```
def santander(hamil):
    def g(w):
        hamil = 2 * w
        print(hamil, w)
        w=hamil
        return hamil
    w=5
    santander = g(w + 1)
    print(w, santander, hamil)
```

Escribe qué se muestra por pantalla cuando se ejecuta la siguiente instrucción:

```
>>>  santander(10)
```

7. Se consideran las siguientes funciones: *(1,25 puntos)*

```
def fun1(a_list, a_cad):
      a_list = ['1', '2', '3']
      for val in a_list:
          a_cad = a_cad + val
      return a_cad

def fun2(a_list, a_cad):
  for c in a_cad:
      if c in a_list:
          a_list.remove(c)
      else:
          a_list.append(c)
```

Escribe qué se muestra por pantalla cuando se ejecutan las siguientes instrucciones:

```
>>> mi_list = ['a', 'b', 'c']
>>> mi_cad = 'abc123'
>>> print(fun1(mi_list, mi_cad))    --------->

>>> print(mi_list)   ----------------->

>>> mi_list = ['1', '2', '3']
>>> my_cad = 'abc123'
>>> print(fun2(mi_list, mi_cad))  -------------->

>>> print(mi_list)   ---------------------->

>>> print(mi_cad)    --------------------->
```

8. En la tabla siguiente $f(n)$ expresa, en milisegundos, el tiempo que un algoritmo tarda en ejecutar un problema de tamaño n. En cada casilla de la tabla se señala el máximo valor que puede tomar n para que un problema de tiempo de resolución $f(n)$ pueda ser resuelto cuando se dispone, respectivamente, de un segundo, un minuto o una hora. Completa la tabla. *(1 punto)*

$f(n)$	1 segundo	1 minuto	1 hora
$\log n$	2^{1000}	2^{60000}	
$\sqrt{n}$	10^6	$36 \cdot 10^8$	
n	10^3	$6 \cdot 10^4$	
n^2	31	244	1897
n^3		39	153
2^n	9	15	21

9. La búsqueda binaria es un algoritmo del tipo ≪divide y vencerás≫ y puede usarse para decidir si un valor dado está o no en una lista ordenada. A continuación se da una ilustración informal recursiva del proceso, aplicado para encontrar un nombre en una guía telefónica asumiendo que hay un nombre por página:

```
Proceso de búsqueda:
     si la guía telefónica tiene una página
               Informamos si el nombre está o no en esa página
      en otro caso
               Partimos la guía telefónica por la mitad
               Aplicamos el proceso  de búsqueda a la parte correspondiente.
```

Desarrolla una función para implementar la búsqueda binaria siguiendo la documentación descrita abajo. No está permitido usar el operador `in`. *(1,5 puntos)*

```
def busqueda_binaria(x, L, i, d):
     ''' Devuelve True si x está en L[i:d+1] y False en otro caso.
     - L es una lista ordenada  de longitud n.
     - i y d son enteros satisfaciendo  0 <= i <= d < n.
     - x es un entero con la propiedad L[i] <= x <= L[d]
     '''
```