Universidad de Alcalá

Nombre y Apellidos: _____

Fundamentos de la Programación

Prueba de Evaluación Extraordinaria Julio 2021 – GIC – GII - GISI

PARTE I – TEORÍA (Duración máxima: 1h)

1. Explica y muestra con un ejemplo la diferencia entre subprogramas que actúan como funciones o como procedimientos. Como ejemplo, esboza una función y un procedimiento que lleven a cabo una tarea similar y haz hincapié en sus similitudes y diferencias, por ejemplo, la forma de denominarlos, los parámetros que reciben y los posibles valores de retorno. **(2 puntos)**

La respuesta a esta pregunta es muy amplia, si bien es esencial mencionar la diferencia fundamental entre ambos tipos de subprograma: que un procedimiento lleva a cabo una tarea en función de una serie de parámetros (si bien puede no recibir parámetros en absoluto) mientras que una función hace también esa tarea a partir de los parámetros y al final retorna un valor, que puede ser único o un conjunto de ellos.

El nombre de la función es comúnmente un sustantivo similar al valor de retorno, así, una función que retorne el cuadrado de un número se llamaría cuadrado. Las funciones lógicas, aquellas que devuelven un valor booleano, suelen nombrarse de modo que denotan un estado utilizando el verbo ser/estar y un valor de comparación (por ejemplo `es_negro()`, `está_vacio()`, `es_primo()`).

Por otra parte, los procedimientos se nombran con verbos que denoten la acción que llevan a cabo, por ejemplo "`mostrar_cuadrado`" que sería un procedimiento que recibe un entero y muestra por pantalla su valor al cuadrado.

```
# función que recibe un número y retorna su cuadrado
def cuadrado(n):
    return n*n

# procedimiento que recibe un número y muestra su cuadrado:
def imprimir_cuadrado(n):
    print n*n
```

En cuanto a la documentación (docstring) los procedimientos reflejan el no retorno `"a, b -> None"` mientras que las funciones reflejan el tipo retornado `"a,b -> c"`

Una diferencia importante también es el modo en que se invocan. La llamada a un procedimiento ocupa una línea completa de código mientras que la invocación a una función se utiliza como cualquier valor del tipo equivalente al tipo del (o los) valores retornados. Así:

```
imprimir_cuadrado(8) #sería una invocación al procedimiento
valor = cuadrado(8) + 15 #invocación a la función
```

2. Al ejecutar el siguiente código, se produce un error. Indica cuál es el motivo de ese error y la forma en la que podríamos evitarlo de manera que se imprimiesen los valores 20 y 10, manteniendo la llamada al subprograma tal y como está. **(1 punto)**

```
def inicializa (a, b):
    a*=20
    b+=10
    return a, b

x,y = inicializa(x,y)
print(x,y)
```

El problema es que se invoca a la función `inicializa()` con dos variables cuyos valores no han sido inicializados. La función los recibe e intenta modificarlos multiplicando `x` por 20 y sumando 10 a `y`. Dado que no tienen un valor aún, no puede multiplicarse por 20 ni sumársele 10 algo que aún no tiene valor y por tanto el código falla.

Para que se obtenga la salida solicitada, habría que inicializar `x` a 1 e `y` a 0, así:

```
x = 1
y = 0
x,y = inicializa(x,y)
print(x,y)
```

3. Estudia el siguiente código e indica qué se mostraría por pantalla al ejecutarlo. Razona tu respuesta **(1 punto)**

```
lista = [1,2,3,[0,0,0]]
lista2 = lista
for i in range(4):
    if i%2 !=0:
        lista2[i] = 'STOP'
print(lista)
```

La salida es:

```
[1, 'STOP', 3, 'STOP']
```

El motivo de esta salida, que podría sorprender inicialmente si uno examina superficialmente el código, es porque `lista` y `lista2` son dos identificadores para una única estructura de datos y por tanto los cambios hechos sobre uno de los identificadores (en este caso en el `for` sobre `lista2`) afectan a la estructura cuyo acceso es común.

4. Estudia el siguiente código e indica cuál sería la salida por pantalla al ejecutarlo **(1 punto)**:

```
def somostar(lista):
    if len(lista)==0:
        x = 0
    else:
        x = somostar(lista[:len(lista)-1])
        print (1 + x)
    return x

somostar([0,2,3,4,5,6])
```

```
1
1
1
1
1
1
```

5. Siguiendo con el código del ejercicio anterior, explica qué ocurriría si la parte "else" de la función la modificásemos de la siguiente manera **(1,5 puntos)**:

```
    else:
        x = lista[len(lista)-1] + somostar(lista[:len(lista)-1])
        # print (1 + x)
    return x
```

y la invocación cambiase a lo siguiente:

```
resultado = somostar([0,2,3,4,5,6])
print(resultado)
```

En este caso, la salida sería:

```
20
```

6. Completa la siguiente función para que retorne la posición del mayor elemento impar de una lista "`lis`" entre las posiciones inicio y fin (ambas incluidas). La función asume que entre las posiciones inicio y fin hay al menos un número impar. **(2 puntos)**

```
def posicion_del_mayor_impar(lis,inicio,fin):
  posicion = inicio
  mayor = lis[posicion]
  for i in range(inicio+1, fin+1):
    if (lis[i] %2 != 0 and lis[i] > mayor):
      mayor = lis[i]
      posicion = i
  return posicion
```

7. Indica la salida del siguiente fragmento de código, teniendo en cuenta que la función split, tal y como se utiliza en este código, transforma la frase original en una lista donde cada palabra es un elemento de la lista: **(1,5 puntos)**

```
frase = "When the night has come and the land is dark".split()
for i in frase:
    b= ""
    for j in range(abs(1 - len(frase))):
      b += frase[j] + ' '
      if len(i) == 2: print(b)
```

¿Variaría dicha salida si la frase hubiera sido "`When the night has come and the land becomes dark`"? Si piensas que sí cambiaría, indica cuál sería la nueva salida.

La salida es:

```
When

When the

When the night

When the night has

When the night has come

When the night has come and

When the night has come and the

When the night has come and the land

When the night has come and the land is
```

Si la frase cambia `is` por `becomes`, no se produciría ninguna salida por la pantalla.

Universidad de Alcalá

Fundamentos de la Programación

Prueba de Evaluación Extraordinaria Julio 2021 – GIC – GII - GISI

PARTE II – PRÁCTICA (Duración máxima: 2h)

E1. Implementa un procedimiento recursivo que reciba un número entero n y devuelva una lista con todos sus divisores incluyendo el 1 y el propio número n. **(2 puntos)**

```
def lista_divisores(n, divisor):
    """ int -> list
        OBJ: Rertorna una lista con los divisores de n, incluyendo 1 y n
        PRE: n >= 0
    """
    if n == 1 or divisor == 1:
        resultado = [1]
    elif n == 0:
        resultado = []
    elif divisor > 1:
        if n % divisor == 0:
            resultado = lista_divisores(n,divisor-1) + [divisor]
        else:
            resultado = lista_divisores(n,divisor-1)
    return resultado

# Prueba:
print(lista_divisores(50,50))
```

E2. Se desea desarrollar un software que permita gestionar información acerca de los vídeos que "ElMorenux", un conocido YouTuber alcalaíno, lleva a cabo en su canal. Concretamente, se te pide lo siguiente:

1. En lo relativo a los datos que permitan almacenar información acerca de un video, nos piden almacenar su título, temática (juegos, viajes, sorteos, etc.), número de *likes*, duración (en minutos) y número de visualizaciones. Para hacer algunas pruebas, genera una variable que contenga 5 vídeos. **(1 punto)**

```
videos = [
    { 'titulo': 'Fornite-especial',
      'tematica': 'juegos',
      'likes': 23000,
      'duracion': 34,
      'visualizaciones': 48399
    },
    { 'titulo': 'LOL-especial',
      'tematica': 'juegos',
      'likes': 67544,
      'duracion': 25,
      'visualizaciones': 200003
    },
    { 'titulo': 'Sorteo-navidad-Play5',
      'tematica': 'sorteo',
```

```
      'likes': 15300,
      'duracion': 56,
      'visualizaciones': 47790
    },
    { 'titulo': 'Andorra-ski',
      'tematica': 'viajes',
      'likes': 278200,
      'duracion': 54,
      'visualizaciones': 348397
    },
    { 'titulo': 'Kun-a-jugar',
      'tematica': 'juegos',
      'likes': 99230,
      'duracion': 24,
      'visualizaciones': 847830
    }
]
```

2. Programa un procedimiento que, a partir de una temática y un rango numérico [menor,mayor], muestre por pantalla la información de todos los vídeos de esa temática que tengan un número de visualizaciones entre esos valores menor/mayor y un mínimo de 1.000 likes. **(1,5 puntos)**

```
def mostrar_tematica_1Klikes(lista_videos,tema,min,max):
    """
       list, str, int, int -> None
       Obj: muestra por pantalla la información de los vídeos de una temática con
            un número de visualizaciones entre min y max, y al menos 1.000 likes
    """
    for video in lista_videos:
        if video['tematica'] == tema \
           and min < video['visualizaciones'] < max \
           and video['likes'] > 1000:
           print(video)
```

3. Programa una función que retorne una lista con todos los vídeos con más de 1 millón de visualizaciones y otra con las que tienen entre medio millón y un millón de visualizaciones. **(2 puntos)**

```
def listas_de_retransmisiones(lista_videos):
    """
       list -> list, list
       Obj: Retorna una lista con los vídeos de más de 1 millón de
            visualizaciones y otra con los de entre medio millón y un millón
    """
    lista_1millon = []
    lista_medio_millon = []
    for video in lista_videos:
        if video['visualizaciones'] > 1000000:
            lista_1millon.append(video)
        elif video['visualizaciones'] > 500000:
            lista_medio_millon.append(video)
    return lista_medio_millon, lista_1millon
```

4. Suponiendo que existe una lista “top_10” con los vídeos más vistos de

ElMorenux, y que dicha lista está ordenada por número de visualizaciones decreciente, programa un método de búsqueda eficiente que permita encontrar un cierto título en concreto. El resultado será la posición en la lista del vídeo buscado, o "None" si el título buscado no se encuentra en dicha lista. **(2 puntos)**

```
# Dado que la lista top_10_videos está ordenada por número de visualizaciones
# no es posible programar un método de búsqueda más eficiente que el secuencial
# para encontrar un cierto título, dado qeu necesitaríamos ordenación por título

def busca_titulo(lista_top_10_videos, titulo):
    """ list, str -> int
        Obj: posición en la lista del vídeo buscado, o "None" si el título buscado no
 se encuentra en dicha lista
    """
    encontrado = False
    pos = None
    i = 0
    while not encontrado and i < len(lista_top_10_videos): # o directamente 10
        if lista_top_10_videos[i]['titulo'] == titulo:
            pos = i
            encontrado = True
        else:
            i += 1
    return pos
```

5. Programa finalmente una función que muestre información estadística sobre la duración (mínima, máxima y media) de aquellos vídeos que cuentan con más de 10.000 likes. **(1,5 puntos)**

```
def mostrar_estadisticas_10Klikes(lista_videos):
    """
       list -> None
       Obj: muestra información estadística sobre la duración (mínima, máxima
            y media) de los vídeos con más de 10.000 likes
    """
    video_min_duracion = video_max_duracion = videos[0]
    duracion_total = num_videos = 0
    for video in lista_videos:
        if video['visualizaciones'] > 10000:
            if video['duracion'] < video_min_duracion['duracion']:
                video_min_duracion = video
            elif video['duracion'] > video_max_duracion['duracion']:
                video_max_duracion = video
            duracion_total += video['duracion']
            num_videos += 1
    if num_videos > 0:
        print('Duracion media de los videos más vistos:', \
                 duracion_total/len(lista_videos))
        print('Video más largo: ', video_max_duracion)
        print('Video más corto: ', video_min_duracion)
    else: print('No hay videos de más de 10.000 likes')
```