Comenzado el	jueves, 17 de diciembre de 2020, 08:26
Estado	Finalizado
Finalizado en	jueves, 17 de diciembre de 2020, 08:59
Tiempo empleado	33 minutos 6 segundos
Puntos	18/25
Calificación	**7** de 10 (**73**%)

Pregunta **1**

Finalizado

Puntúa 1 sobre 1

El producto de dos números *a* y *b* mayores o iguales cero, es en última instancia *una suma*: se puede ver como sumar *b* veces el número *a*, o como sumar *a* veces el número *b*. Por ejemplo: 5 * 3 = 5 + 5 + 5 o bien 5 * 3 = 3 + 3 + 3 + 3 + 3. Sabiendo esto, se puede intentar hacer una definición recursiva de la operación producto(a, b) [a >= 0, b >= 0], que podría ser la que sigue:

$$\textbf{producto(a, b)} = \begin{cases} 0 & \text{si } a == 0 \text{ o } b == 0 \\ a + \text{producto}(a, b-1) & \text{si } a > 0 \text{ y } b > 0 \end{cases}$$

[a y b enteros, a >= 0 y b >= 0]

¿Es correcto el siguiente planteo de la función *producto(a, b)*?

```
def producto(a, b):
    if a == 0 or b == 0:
        return 0
    return a + producto(a, b-1)
```

Seleccione una:

a.
Sí. Es correcto.

b.
No. La última línea debería decir *return a + producto(a-1, b-1)* en lugar de *return a + producto(a, b-1)*.

c.
No. La función sugerida siempre retornará 0, sean cuales fuesen los valores de *a* y *b*.

d.
No. La propia definición previa del concepto de producto es incorrecta: un producto es una multiplicación, y no una suma...

Pregunta **2**

Finalizado

Puntúa 1 sobre 1

Suponga que se desea desarrollar un programa que cargue dos números, y muestre el mayor de los números pero también el valor de multiplicar por dos y por tres a ese mayor. ¿Está bien planteado el programa que sigue?

```
__author__ = 'Cátedra de AED'


def cargar():
    a = int(input('A: '))
    b = int(input('B: '))
    return a, b


def comparar(a, b):
    if a > b:
        may = a
    else:
        may = b
    return may


def procesar(may):
    doble = 2 * may
    triple = 3 * may
    return doble, triple


def mostrar(may, doble, triple):
    print('El mayor es:', may)
    print('El doble del mayor es:',
doble)
    print('El triple del mayor es:',
triple)


# script principal...
a, b = cargar()
doble, triple = procesar(may)
may = comparar(a, b)
mostrar(may, doble, triple)
```

Seleccione una:

a.
El programa está mal planteado: la función *comparar()* debería ser invocada antes de invocar a la función *procesar()*. Así como está, lanza un error al intentar ejecutar la función *procesar()* pues no reconoce a la variable may.

b.
El programa lanza un error de intérprete: el script principal no puede consistir sólo en llamadas a funciones.

c.
El programa está mal planteado: la visualización de los resultados finales DEBE hacerse en el script principal y NO en una función separada.

d.
Sí. El programa está correctamente planteado.

Pregunta **3**

Finalizado

Puntúa 0 sobre 1

¿Cuál de las siguientes ***es cierta*** respecto del uso y aplicación del método *seek()* en archivos de texto con Python?

Seleccione una:

a.
Si el archivo es de texto, en Python el método *seek()* sólo puede usarse para cambiar el valor del *file pointer* suponiendo que el reposicionamiento se hace desde el inicio del archivo (segundo parámetro igual a *io.SEEK_SET = 0*). La única excepción es la de reposicionar el *file pointer* al final del archivo (*m.seek(0, io.SEEK_END)*)

b.
Tanto da que el archivo sea de texto o binario, en Python el método *seek()* se aplica exactamente en la misma forma, sin restricciones.

c.
Si el archivo es de texto, en Python el método *seek()* puede usarse para cambiar el valor del *file pointer* suponiendo que el reposicionamiento se hace desde el inicio del archivo (segundo parámetro igual a *io.SEEK_SET = 0*) o desde la posición actual del file pointer (segundo parámetro igual a *io.SEEK_CUR = 1*), sin restricciones en ambos casos.

d.
Si el archivo es de texto, en Python el método *seek()* puede usarse para cambiar el valor del *file pointer* suponiendo que el reposicionamiento se hace desde el inicio del archivo (segundo parámetro igual a *io.SEEK_SET = 0*) o desde el final del archivo (segundo parámetro igual a *io.SEEK_SET = 2*), sin restricciones en ambos casos.

Pregunta **4**

Finalizado

Puntúa 1 sobre 1

¿Cuál de las siguientes expresiones describe la forma de trabajo general de una *Cola*?

Seleccione una:

a.
LIFO (Last In - First Out)

b.
OIRO (Ordered In - Random Out)

c.
FIFO (First In - First Out)

d.
OIFO (Ordered In - First Out)

Pregunta **5**

Finalizado

Puntúa 1 sobre 1

*¿Cuáles de las siguientes afirmaciones **serían ciertas** si el problema **P** vs NP se resolviese por la negativa (es decir, si se demostrase que P y NP **no son** realmente iguales*? (Más de una respuesta puede ser válida... marque todas las que considere correctas...)

Seleccione una o más de una:

a.
Se tendría la certeza de que al menos para algunos problemas no existen buenas soluciones, lo que permitiría dejar de buscarlas inútilmente y pasar a concentrar los esfuezos en diseñar soluciones aproximadas o no óptimas.

b.
Todo problema actualmente considerado intratable tendría solución de tiempo de ejecución polinomial en una máquina determinista.

c.
Todo problema actualmente en NP, tendría solución de tiempo polinomial en una máquina determinista.

d.
Todo problema actualmente en P se convertiría en intratable.

Pregunta **6**

Finalizado

Puntúa 1 sobre 1

Considere el problema del cálculo de los montos a pagar por el viaje de estudios de tres cursos de secundaria, tal como se presentó en la Ficha 10. El programa que se muestra más abajo, está replanteado de forma de usar variables globales (**ver Ficha 09**) en lugar de parámetros y retornos, y la función *mayor()* (que determinaba cuál era el curso con más cantidad de alumnos, y además, cuál era el monto a pagar por ese curso), está definida de la siguiente forma:

```
__author__ = 'Cátedra de AED'


def mayor():
    global may, may_cur
    if c1 > c2 and c1 > c3:
        may = m1
        may_cur = 'Primero'
    else:
        if c2 > c3:
            may = m2;
            may_cur = 'Segundo'
        else:
            may = m3
            may_cur = 'Tercero'


def montos():
    global m1, m2, m3
    m1 = c1 * 1360
    m2 = c2 * 1360
    m3 = c3 * 1360

    if c1 > 40:
        m1 = m1 - m1/100*5

    if c2 > 40:
        m2 = m2 - m2/100*5

    if c3 > 40:
        m3 = m3 - m3/100*5


def porcentaje():
    global mtot, porc
    mtot = m1 + m2 + m3
    if mtot != 0:
        porc = may / mtot * 100
    else:
        porc = 0


def test():
    global c1, c2, c3

    # título general y carga de datos...
    print('Cálculo de los montos de un viaje de
estudios...')
    c1 = int(input('Ingrese la cantidad de alumnos
del primer curso:  '))
    c2 = int(input('Ingrese la cantidad de alumnos
del segundo curso:  '))
    c3 = int(input('Ingrese la cantidad de alumnos
del tercer curso:  '))

    # procesos... invocar a las funciones en el
orden correcto...
    montos()
    mayor()
    porcentaje()

    # visualización de resultados
    print('El curso mas numeroso es el', may_cur)
    print('El monto del viaje del primer curso
es:', m1)
    print('El monto del viaje del segundo curso
es:', m2)
    print('El monto del viaje del segundo curso
es:', m3)
    print('El porcentaje del monto del mas
numeroso en el total es:', porc)


# script principal: sólo invocar a test()...
test()
```

¿Qué efecto causaría en el programa que la función *mayor()* fuese reemplazada por esta otra versión que se muestra a continuación, y qué cambios debería hacer el programador para que su programa siga

funcionando y cumpliendo con los mismos requerimientos? (Más de una respuesta puede ser válida. Marque todas las que considere correctas):

```
def mayor():
    global salida
    if c1 > c2 and c1 > c3:
        salida = 'Primero', m1
    else:
        if c2 > c3:
            salida = 'Segundo', m2
        else:
            salida = 'Tercero', m3
```

Seleccione una o más de una:

a.

En lugar de usar dos variables simples para almacenar los dos resultados, usa la tupla *salida* para guardarlos a ambos. Un cambio que debe hacer el programador es en la visualización de resultados dentro de la función *test()*, que debería verse así:

```
# función test()... visualización de
resultados...
print('El curso mas numeroso es el',
salida[1])
print('El monto del viaje del primer curso
es:', m1)
print('El monto del viaje del segundo curso
es:', m2)
print('El monto del viaje del segundo curso
es:', m3)
print('El porcentaje del monto del mas
numeroso en el total es:', porc)
```

b.

En lugar de usar dos variables simples para almacenar los dos resultados, usa la tupla *salida* para guardarlos a ambos. Un cambio que debe hacer el programador es en la visualización de resultados dentro de la función *test()*, que debería verse así:

```
# función test()... visualización de
resultados...
print('El curso mas numeroso es el',
salida[0])
print('El monto del viaje del primer curso
es:', m1)
print('El monto del viaje del segundo curso
es:', m2)
print('El monto del viaje del segundo curso
es:', m3)
print('El porcentaje del monto del mas
numeroso en el total es:', porc)
```

c.

En lugar de usar dos variables simples para almacenar los dos resultados, usa la tupla *salida* para guardarlos a ambos. Un cambio que debe hacer el programador es en la función *porcentaje()*, que debería verse así:

```
def porcentaje():
    global mtot, porc
    mtot = m1 + m2 + m3
    if mtot != 0:
        porc = salida[1] / mtot * 100
    else:
        porc = 0
```

d.

En lugar de usar dos variables simples para almacenar los dos resultados, usa la tupla *salida* para guardarlos a ambos. Un cambio que debe hacer el programador es en la función *porcentaje()*, que debería verse así:

```
def porcentaje():
    global mtot, porc
    mtot = m1 + m2 + m3
    if mtot != 0:
        porc = salida[0] / mtot * 100
    else:
        porc = 0
```

Pregunta **7**

Finalizado

Puntúa 1 sobre 1

¿Qué significa decir que un algoritmo dado tiene un tiempo de ejecución **O(n)**?

Seleccione una:

a.
El tiempo de ejecucíón es constante, sin importar la cantidad de datos.

b.
A medida que aumenta el número de datos, aumenta el tiempo pero en forma muy suave: el conjunto de datos se divide en dos. se procesa una de las mitades, se desecha la otra y se repite el proceso hasta que no pueda volver a dividirse la mitad que haya quedado.

c.
El tiempo de ejecución es lineal: si aumenta el número de datos, aumenta el tiempo en la misma proporción.

d.
El proceso normalmente consiste en dos ciclos (uno dentro del otro) de aproximadamente n iteraciones cada uno, de forma que las operaciones críticas se aplican un número cuadrático de veces.

Pregunta **8**

Finalizado

Puntúa 1 sobre 1

¿Cuál de los siguientes scripts ***NO*** creará una tupla conteniendo exclusivamente los caracteres de la palabra 'Python', en ese mismo orden'? (Repetimos: la pregunta es cuál de todos ***NO*** creará la tupla pedida...)

Seleccione una:

a.
```
t2 = 'P', 'y', 't', 'h', 'o', 'n'
```

b.
```
t3 = ('P', 'y', 't', 'h', 'o', 'n')
```

c.
```
t1 = tuple('Python')
```

d.
```
Python = 'Java'
t5 = tuple(Python)
```

e.
```
t4 = ()
for c in 'Python':
     t4 += c,
```

Pregunta **9**

Finalizado

Puntúa 1 sobre 1

Analice el siguiente script simple, cuyo objetivo es tomar por teclado los datos básicos de un postulante a un crédito, y mostrar por consola estándar los datos cargados:

```
a = input('Ingrese su nombre: ')
print('El nombre ingresado es: ', a)
a = int(input('Ahora ingrese su edad: '))
print('La edad ingresada es: ', a)
a = float( input('Y ahora ingrese sueldo: ') )
print('El sueldo ingresado es: ', a)
```

¿Producirá algún problema la ejecución de este script?

Seleccione una:

a.
Sí. El script ejecutará sin problemas aparentes, pero mostrará en consola estándar siempre el mismo valor: el valor *None*.

b.
Sí. El script comenzará a ejecutarse, pero lanzará un error y se interrumpirá cuando intente cargar la edad del postulante en la variable a que ya contenía el nombre.

c.
No. El script se ejecutará sin problemas y hará lo esperado.

d.
Sí. El script ejecutará sin problemas aparentes, pero mostrará en consola estándar siempre el mismo valor: el nombre del postulante.

Pregunta **10**

Finalizado

Puntúa 0 sobre 1

¿Cuál de los siguientes es el inconveniente principal que supone procesar el *mismo archivo de texto* en *diferentes plataformas* (o sistemas operativos)?

Seleccione una:

a.
La forma en que se trata y reconoce el final del archivo. Cada plataforma hace esto de forma diferente en un archivo de texto, y con cada lenguaje de programación debe considerarse este problema en forma particular.

b.
El tratamiento dado al *salto de linea*: en algunas plataformas se representa con un caracter de control simple (\n) y en otras con una pareja (\r\n). En general, los lenguajes de programación modernos (como Python) resuelven automáticamente este inconveniente.

c.
La forma en que se distinguen las mayúsculas de las minúsculas dentro del archivo.

d.
No hay ningún inconveniente especial. El mismo archivo de texto puede abrirse y manejarse sin problema alguno en cualquier plataforma, sin tener que tomar precauciones particulares.

Pregunta **11**

Finalizado

Puntúa 1 sobre 1

¿Cuáles de las siguientes afirmaciones son correctas respecto del algoritmo de backtracking que se propuso en clases para el problema de las Ocho Reinas (que mostramos más abajo)? (Más de una respuesta puede ser cierta, por lo que marque todas las que considere correctas).

```
def intend(col):
    global rc, qr, qid, qnd

    fil, res = -1, False
    while not res and fil != 7:
        res = False
        fil += 1
        di = col + fil
        dn = (col - fil) + 7
        if qr[fil] and qid[di] and qnd[dn]:
            rc[col] = fil
            qr[fil] = qid[di] = qnd[dn] = False

            if col < 7:
                res = intend(col + 1)
                if not res:
                    qr[fil] = qid[di] = qnd[dn] = True
                    rc[col] = -1
            else:
                res = True
    return res
```

Seleccione una o más de una:

- [ ] a.
  Una invocación de la forma intend(5) significa que se va a intentar posicionar en alguna fila a la reina de la columna 5.

- [x] b.
  Para mostrar todas las posibles soluciones básicas o simétricas, básicamente solo hay que cambiar el ciclo while y la bandera usada para cortar al encontrar una solución, por un for que obligatoriamente recorra las ocho filas en cada una de las ocho columnas.

- [ ] c.
  El algoritmo de resolución del problema nunca analiza las diagonales del tablero (solo las filas y las columnas), y de allí deduce las soluciones correctas.

- [ ] d.
  Se usan cuatro arreglos unidimensionales para llevar el control de las casillas disponibles y un quinto arreglo para representar la solución.

Pregunta **12**

Finalizado

Puntúa 1 sobre 1

Asuma que se quiere definir una función *registrar()* como la que se muestra más abajo, en la cual se necesita crear una matriz *reg* de forma que en esa matriz cada fila represente un año calendario desde el 2000 en adelante (la fila 0 representaría al año 2000, la 1 al 2001 y así sucesivamente) y cada columna represente una de las carreras disponibles en una universidad (numeradas desde 0 en adelante). La idea es que una vez creada esa matriz, debe usarse para almacenar en cada casillero una cierta cantidad fija de inscriptos ¿Hay algún problema con la siguiente función en Python, asumiendo que ***a*** es la cantidad de filas y ***c*** es la cantidad de columnas?

```
def registrar(a, c):
    reg = []
    for i in range(a):
        for j in range(c):
            reg[i][j] = 50

    return reg
```

Seleccione una:

a.
Sí, hay un problema: los índices *i* y *j* están al revés cuando se usaron como índices para entrar a una casilla de la matriz.

b.
Sí, hay un problema pero es menor: la función crea efectivamente la matriz pedida, pero luego inicializa cada casillero con el mismo valor, lo cual no tiene sentido práctico.

c.
Sí, hay un problema: al ejecutar la función de producirá un error de intérprete y el programa se interrumpirá, debido a que la matriz en realidad nunca fue creada.

d.
No, no hay problema alguno. La función crea la matriz pedida, e inicializa cada casillero con el valor 50.

Pregunta **13**

Finalizado

Puntúa 0 sobre 1

¿Cuáles de las siguientes son CIERTAS en relación al sistema de coordenadas de pantalla?

Seleccione una o más de una:

a.
El origen del sistema de coordenadas se encuentra en el punto inferior izquierdo de la pantalla o de la ventana que se esté usando.

b.
Las filas de pantalla o de ventana con número de orden más bajo, se encuentran más cerca del borde superior que las filas con número de orden más alto.

c.
Las filas de pantalla o de ventana con número de orden más alto, se encuentran más cerca del borde superior que las filas con número de orden más bajo.

d.
El origen del sistema de coordenadas se encuentra en el punto superior izquierdo de la pantalla o de la ventana que se esté usando.

Pregunta **14**

Finalizado

Puntúa 0 sobre 1

¿Cuál es la cantidad de niveles del *árbol de invocaciones recursivas* que se genera al ejecutar el *Quicksort* para ordenar un arreglo de *n* elementos, en el ***peor caso***? (Es decir: ¿Cuál es la *altura* de ese árbol en ese peor caso?)

Seleccione una:

a.
`Altura = n`

b.
`Altura = log(n)`

c.
`Altura = n`^{2}

d.
`Altura = n * log(n)`

Pregunta **15**

Finalizado

Puntúa 1 sobre 1

A lo largo del curso hemos visto la forma de plantear programas controlados por menú de opciones, y sabemos que esos programas incluyen un ciclo para el control del proceso. Pero también podríamos hacer un planteo basado en recursión, sin usar el ciclo, en forma similar a la que se muestra aquí:

```
__author__ = 'Cátedra de AED'


def menu():
    print('1. Opcion 1')
    print('2. Opcion 2')
    print('3. Salir')
    op = int(input('Ingrese opcion: '))
    if op == 1:
        print('Eligio la opcion 1...')
    elif op == 2:
        print('Eligio la opcion 2...')
    elif op == 3:
        return
    menu()


# script principal...
menu()
```

¿Hay algún problema o contraindicación respecto del uso y aplicación de esta versión recursiva para la gestión de un menú? Seleccione la respuesta que mejor describa este punto.

Seleccione una:

a.
Esta versión recursiva posiblemente sea más simple y compacta que la versión basada en un ciclo, pero la versión recursiva pide memoria de stack cada vez que se invoca, por lo que es más conveniente la iterativa.

b.
Esta versión recursiva es completamente equivalente a la versión iterativa. No hay ningún motivo para preferir la una sobre la otra. Da lo mismo si el programador usa la recursiva o la iterativa.

c.
La versión recursiva que se mostró no funciona. Sólo permite cargar una vez la opción elegida, e inmediatamente se detiene.

d.
No hay ningún problema ni contraindicación. Y no sólo eso: la versión recursiva es más simple y compacta que la versión basada en un ciclo, por lo cual es incluso preferible usarla.

Pregunta **16**

Finalizado

Puntúa 1 sobre 1

En la columna de la izquierda se enumeran algunas situaciones típicas de programación. Seleccione de la columna de la derecha cuál es el tipo de ciclo que sería más adecuado o cómodo para usar en cada caso (sin que esto implique que el ciclo elegido sea obligatorio para cada caso...):

Esquema en el cual se sabe de antemano el número de repeticiones	Ciclo for (iterando sobre un range cuyo tamaño coincida con la c
Recorrido de una secuencia (tupla, range, cadena, lista, etc.)	Ciclo for (iterando sobre la estructura de datos a procesar)
Esquema en el cual se desconoce de antemano el número de repeticiones	Ciclo while (operando en forma natural como [0, n])
Programas controlados por menú	Ciclo do while (forzado a la forma [1, N] y conteniendo un if anid
Validación de valores cargados por teclado.	Ciclo while (forzado a la forma [1, N] y que incluya una condición

Pregunta **17**

Finalizado

Puntúa 1 sobre 1

¿Cuáles de los siguientes son *factores de eficiencia comunes* a considerar en el análisis de algoritmos? (Más de una respuesta puede ser válida... marque *todas* las que considere correctas).

Seleccione una o más de una:

- a. El tiempo de ejecución.
- b. El consumo de memoria.
- c. La calidad aparente de la interfaz de usuario.
- d. La complejidad aparente del código fuente.

Pregunta **18**

Finalizado

Puntúa 1 sobre 1

Otra vez suponga que se ha creado y cargado por teclado un arreglo *peaje* en el que almacenaron cadenas de caracteres que representan las patentes de los vehículos que se registraron en una estación de peaje en un período dado. Suponga que se pide desarrollar una función que determine cuantas veces está registrada en el arreglo *peaje* la patente cargada en la variable *pat*. Suponga que se propone para cumplir esa tarea la función *search()* que se muestra más abajo, con la instrucción break que se ve en rojo al final de la rama vervadera de if dentro del ciclo ¿Qué puede decirse respecto del planteo de esa función *search()*?

```
def search(peaje, pat):
    c = 0
    for p in peaje:
        if p == pat:
            c += 1
            break

    if(c != 0):
        print('Cantidad de pasos registrados para ese vehiculo:', c)
    else:
        print('No esta registrado ese vehiculo')
```

Seleccione una:

a.
La función ya no cumplirá con el objetivo: si el vector peaje tuviese efectivamente más de una vez un valor igual a *pat*, contaría sólo el primero de ellos y detendría el ciclo. Así planteada, la función está haciendo una búsqueda secuencial de primera ocurrencia, en lugar de un conteo de todas las ocurrencias.

b.
La función contará todas las ocurrencias del valor *pat*, salvo la primera.

c.
La combinación entre la instrucción *break* y el contador *c* que aparece dentro del ciclo, provoca que la función cuente sólo algunas de las ocurrencias del valor *pat*: sólo aquellas que se encuentren en casillas con índice par (los de las casillas 0, 2, 4, 6, etc.)

d.
El programa funcionará correctamente de todos modos, y contará todas las repeticiones del valor *pat*.

Pregunta **19**

Finalizado

Puntúa 1 sobre 1

¿Cuáles de las siguientes afirmaciones ***son ciertas*** en relación a conceptos asociados con la *recursividad*?

Seleccione una o más de una:

- a.
  Si una función es recursiva, entonces no debe incluir ningún ciclo en su bloque de acciones.

- b.
  Cada instancia recursiva que es ejecutada, almacena dos grupos de datos en el stack segment: la dirección de retorno (a la que se debe regresar cuando termine la ejecución de esa instancia) y las variables locales que esa instancia de la función haya creado.

- c.
  A medida que se desarrolla la cascada de invocaciones recursivas, el stack segment se va llenando para darle soporte a cada instancia recursiva, y luego, cuando una instancia logra finalizar y se produce el proceso de vuelta atrás, el stack segment comienza a vaciarse.

- d.
  Una función recursiva no puede incluir más de una invocación a si misma en su bloque de acciones.

Pregunta **20**

Finalizado

Puntúa 1 sobre 1

Suponga que se ha creado y cargado por teclado un arreglo *peaje* en el que almacenaron cadenas de caracteres que representan las patentes de los vehículos que se registraron en una estación de peaje en un período dado. Suponga que se pide desarrollar una función que determine cuantas veces está registrada en el arreglo *peaje* la patente cargada en la variable *pat*. Suponga que se propone para cumplir esa tarea la función *search()* que se muestra más abajo, con la rama else que se ve en rojo como parte del if dentro del ciclo. ¿Qué puede decirse respecto del planteo de esa función *search()*?

```
def search(peaje, pat):
    c = 0
    for p in peaje:
        if p == pat:
            c += 1
        else:
            print('No esta registrado ese vehiculo')

    if(c != 0):
        print('Cantidad de pasos registrados para ese vehiculo:', c)
```

Seleccione una:

a.
La función sólo contará una vez la patente indicada por *pat.*

b.
El planteo ***es incorrecto***: cada vez que se compruebe una casilla que NO tenga la patente buscada, aparecerá el mensaje indicando que no existe (aún cuando esa patente podría existir una o muchas veces).

c.
El programa funcionará correctam*ente* sin ningún inconveniente.

d.
La combinación entre el *else* y el contador *c* que aparece dentro del ciclo, provoca que la función cuente sólo algunas de las patentes cuyo valor coincida con *pat*: sólo aquellos que se encuentren en casillas con índice par (los de las casillas 0, 2, 4, 6, etc.)

Pregunta **21**

Finalizado

Puntúa 1 sobre 1

Se tiene un algoritmo que realiza cierta cantidad de procesos sobre un conjunto de ***n*** datos y un minucioso análisis matemático ha determinado que la cantidad de procesos que el algoritmo realiza en el peor caso viene descripto por la función $f(n) = 3n^3 + 5n^2 + 2n^{1.5}$ ¿Cuál de las siguientes expresiones representa mejor el orden del algoritmo para el peor caso?

Seleccione una:

a.
$O(3n^3 + 5n^2 + 2n^{1.5})$

b.
$O(n^{1.5})$

c.
$O(n^3 + n^2)$

d.
$O(n^3)$

Pregunta **22**

Finalizado

Puntúa 1 sobre 1

¿Cuál de las siguientes afirmaciones es ***incorrecta*** en relación al concepto de *módulo* en Python, y/o a elementos asociados al uso de módulos en Python?

Seleccione una:

a.
Cuando se usa una instrucción import, Python busca primero si existe un módulo estándar cuyo nombre coincida con el del módulo importado. Si no lo encuentra, busca entonces en la lista de carpetas indicada por la variable sys.path.

b.
Un módulo en Python puede tener elementos *docstring* que documenten su uso y su contenido.

c.
Cada vez que un módulo es importado en un programa, su contenido es vuelto a compilar por el intérprete.

d.
Un módulo en Python es cualquier archivo con extensión *.py* (código fuente que contenga funciones, definiciones generales, clases, variables, etc.)

e.
Un módulo siempre puede ser importado desde otro módulo, mediante instrucciones *import* o *from import*.

f.
Un módulo pensado para *ser ejecutado en forma directa* debería contener un script de control de la forma *if __name__ == '__main__'* para evitar que al ser importado se ejecute en forma inconveniente su posible script principal.

Pregunta **23**

Finalizado

Puntúa 0 sobre 1

¿Cuál de las siguientes expone correctamente la estrategia general que debe llevar a cabo un *proceso de baja lógica de registros en un archivo*?

Seleccione una:

a.
1) Abrir el archivo *m* en modo 'r+b'. 2) Determinar si el archivo contiene o no a un registro con la misma clave que se quiere eliminar. 3) Si no existe un registro con esa clave, rechazar la baja. Si existe, asegurarse de marcar el registro como *eliminado* (*activo = False*), y volver a grabar el registro en la misma posición que tenía en el arrchivo.

b.
1) Abrir el archivo *m* en modo 'a+b'. 2) Determinar si el archivo contiene o no a un registro con la misma clave que se quiere eliminar. 3) Si no existe un registro con esa clave, rechazar la baja. Si existe, asegurarse de marcar el registro como *eliminado* (*activo = False*), y volver a grabar el registro en la misma posición que tenía en el archivo.

c.
1) Abrir el archivo *m* en modo 'r+b'. 2) Determinar si el archivo contiene o no a un registro con la misma clave que se quiere eliminar. 3) Si no existe un registro con esa clave, rechazar la baja. Si existe, asegurarse de marcar el registro como *no eliminado* (*activo = True*), y volver a grabar el registro en la misma posición que tenía en el archivo.

d.
1) Abrir el archivo *m* en modo 'rb'. 2) Determinar si el archivo contiene o no a un registro con la misma clave que se quiere eliminar. 3) Si no existe un registro con esa clave, rechazar la baja. Si existe, asegurarse de marcar el registro como *eliminado* (*activo = False*), y volver a grabar el registro en la misma posición que tenía en el archivo.

Pregunta **24**

Finalizado

Puntúa 1 sobre 1

¿Cuáles de las siguientes afirmaciones con correctas en relación a conceptos elementales del análisis de algoritmos? (Más de una respuesta puede ser válida, por lo que marque todas las que considere correctas).

Seleccione una o más de una:

- a. El análisis del *caso promedio* es aquel en el cual se estudia el comportamiento de un algoritmo cuando debe procesar una configuración de datos que llegan en forma aleatoria.
- b. La notación Big O se usa para expresar el rendimiento de un algoritmo en terminos de una función que imponga una cota inferior para ese algoritmo en cuanto al factor medido (tiempo o espacio de memoria).
- c. El análisis del *peor caso* es aquel en el cual se estudia el comportamiento de un algoritmo cuando debe procesar la configuración más desfavorable posible de los datos que recibe.
- d. Los dos factores de eficiencia más comúnmente utilizados en el análisis de algoritmos son el tiempo de ejecución de un algoritmo y el espacio de memoria que un algoritmo emplea.

Pregunta **25**

Finalizado

Puntúa 1 sobre 1

¿Cuáles de los siguientes problemas pertenecen a la clase de complejidad NP (pero no a la clase de complejidad P (según lo que hasta hoy se sabe)? (Suponga que en todos los casos se está haciendo referencia a la versión *de decisión* del problema indicado) (Más de una respuesta puede ser válida. Marque todas las que considere correctas).

Seleccione una o más de una:

- a. El problema de la Inserción de un nuevo elemento en una pila de n componentes.
- b. El problema del Clique Máximo en un grafo no dirigido con n vértices.
- c. El problema de la Satisfactibilidad Booleana.
- d. El problema de la Multiplicación de Matrices (para simplificar, suponga matrices cuadradas y del mismo orden n).

◄ Avisos

Ir a...