| | |
|---|---|
| **Comenzado el** | domingo, 27 de mayo de 2018, 16:34 |
| **Estado** | Finalizado |
| **Finalizado en** | domingo, 27 de mayo de 2018, 17:07 |
| **Tiempo empleado** | 32 minutos 51 segundos |
| **Puntos** | 22/22 |
| **Calificación** | **10** de 10 (**100**%) |

**Pregunta 1**

Correcta

Puntúa 1 sobre 1

¿Cuál de las siguientes es claramente ***FALSA*** respecto de la estrategia de dividir un problema en subproblemas?

Seleccione una:

a.
La división en subproblemas garantiza que se encontrará una solución para el problema en estudio. ✓

¡Ok! Lamentablemente, nada garantiza que encontremos una solución para cualquier problema que se nos plantee...

b.
La división en subproblemas permite generar subrutinas que luego podrán usarse nuevamente en otros planteos.

c.
La división en subproblemas facilita la comprensión del problema por parte del programador.

d.
La división en subproblemas permite dividir el trabajo entre varios programadores.

¡Correcto!

La respuesta correcta es:
La división en subproblemas garantiza que se encontrará una solución para el problema en estudio.

**Pregunta 2**

Correcta

Puntúa 1 sobre 1

¿Qué hace el siguiente programa en Python?

```
__author__ = 'Cátedra de AED'


def comparar():
    if a > p:
        print('El valor', a, 'es mayor...' )

    if b > p:
        print('El valor', b, 'es mayor...')

    if c > p:
        print('El valor', c, 'es mayor...')


def promedio():
    global p
    p = (a + b + c) / 3


a = int(input('A: '))
b = int(input('B: '))
c = int(input('C: '))

promedio()
comparar()

print('Programa terminado...')
```

Seleccione una:

a.
Calcula el promedio de los valores de *a*, *b* y *c*, y luego muestra sólo los valores de de las variables que sean mayores al promedio. ✓

¡Ok!

b.
Lanza error de intérprete: las variables *a*, *b*, *c* y *p* están definidas incorrectamente (debieron definirse todas en el script principal, debajo de las funciones).

c.
Muestra los valores de *a*, *b*, *c*, y también el *promedio* de los valores de esas tres variables.

d.
Calcula el promedio de los valores de *a*, *b* y *c*, y luego muestra los valores de todas las variables.

¡Correcto!

La respuesta correcta es:
Calcula el promedio de los valores de *a*, *b* y *c*, y luego muestra sólo los valores de de las variables que sean mayores al promedio.

**Pregunta 3**

Correcta

Puntúa 1 sobre 1

En la columna de la izquierda se describen algunas situaciones de programación que necesitan ser resueltas mediante el desarrollo de alguna función, y se muestra el bloque de acciones de las posibles funciones, sin sus cabeceras. Y en la columna de la derecha, figuran las posibles cabeceras de ese funciones. Para cada situación, seleccione la cabecera que sería adecuada.

Tomar como parámetro una cadena de caracteres y dos números enteros. Retornar los dos caracteres de la cadena que estén ubicados en las posiciones indicadas por los dos números:

def valores(s, i1, i2 ) ▾

✓

```
# Cabecera??
  return s[i1], s[i2]
```

Tomar como parámetro los valores de dos temperaturas, y retornar un valor lógico que indique si ambas son mayores a cero:

```
# Cabecera??
```

def chequear(t1, t2): ▾

✓

```
if t1 > 0 and t2 > 0:
    r = True
else:
    r = False
return r
```

Tomar como parámetro dos números y un valor boolean. Si el boolean es true, calcular y retornar el cociente de los dos números. Si el boolean es false, retornar el producto de ambos números:

def proceso(x, y, flag): ▾

```
# Cabecera??
```

✓

```
if flag:
    r = x / y
else:
    r = x * y
return r
```

Tomar como parámetro las edades de tres personas, y retornar la menor edad:

```
# Cabecera??
```

```
if e1 < e2 and e1 < e3:
    m = e1
else:
    if e2 < e3:
        m = e2
    else:
        m = e3
return m
```

def menor(e1, e2, e3): ▾

✓

¡Ok!

La respuesta correcta es:

Tomar como parámetro una cadena de caracteres y dos números enteros. Retornar los dos caracteres de la cadena que estén ubicados en las posiciones indicadas por los dos números:

```
# Cabecera??
  return s[i1], s[i2]
```

→ def valores(s, i1, i2 ),

Tomar como parámetro los valores de dos temperaturas, y retornar un valor lógico que indique si ambas son mayores a cero:

```
# Cabecera??

  if t1 > 0 and t2 > 0:
      r = True
  else:
      r = False
  return r
```

→ def chequear(t1, t2):,

Tomar como parámetro dos números y un valor boolean. Si el boolean es true, calcular y retornar el cociente de los dos números. Si el boolean es false, retornar el producto de ambos números:

```
# Cabecera??

  if flag:
      r = x / y
  else:
      r = x * y
  return r
```

→ def proceso(x, y, flag):,

Tomar como parámetro las edades de tres personas, y retornar la menor edad:

```
# Cabecera??
   if e1 < e2 and e1 < e3:
       m = e1
   else:
       if e2 < e3:
           m = e2
       else:
           m = e3
   return m
```

→ def menor(e1, e2, e3):

**Pregunta 4**

Correcta

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:', d
oble)
    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.
Sí. El programa está correctamente planteado.

b.
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. ✓

¡Ok!

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

¡Correcto!

La respuesta correcta es:
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.

**Pregunta 5**

Correcta

Puntúa 1 sobre 1

Suponga que se quiere desarrollar un programa que cargue por teclado el nombre de una persona, su edad, y su sueldo anual promedio, y que luego se desea invocar a una función llamada **convertir()** que tome como parámetros a esos valores, los procese, y retorne una cadena con todos esos datos convertidos a una cadena de caracteres y concatenados. Tomando como modelo de uso a la función *test()* que se muestra en el programa de mas abajo... ¿cuál de las que se proponen como opciones podría ser la cabecera de la función **convertir()**?

```
__author__ = 'Cátedra de AED'


# Suponga que aquí está definida la función convertir
()...
# def .....
#   ............
#   ............


def test():
    nombre = input('Ingrese el nombre: ')
    edad = int(input('Ingrese edad: '))
    sueldo = float(input('Ingrese sueldo:'))

    resultado = convertir(nombre, edad, sueldo)

    print('Datos registrados:', resultado)


# script principal
test()
```

Seleccione una:

a.
def convierta(nombre, edad, sueldo):

b.
def convertir():

c.
def convertir(nombre, edad, sueldo): ✓

¡Ok!

d.
def convertir(edad, sueldo):

¡Correcto!

La respuesta correcta es:
def convertir(nombre, edad, sueldo):

**Pregunta 6**

Correcta

Puntúa 1 sobre 1

Suponga que se quiere desarrollar un programa para cargar una cadena de caracteres por teclado y mostrar esa cadena en la consola de salida mediante una función ¿Cuál de las posibles respuestas *describe mejor* lo que ocurre con el programa que mostramos aquí?

```
__author__ = 'Cátedra de AED'


def mensaje(m):
    print('Mensaje:', m)


def test():
    mens = input('Ingrese un mensaje a mostrar: ')
    r = mensaje(mens)
    print('Programa terminado...')

# script principal...
test()
```

Seleccione una:

a.
El programa compila (el intérprete no lanza error alguno al comenzar a ejecutarlo) y ejecuta, pero no muestra correctamente el mensaje esperado: en su lugar, muestra el valor *None*.

b.
El programa no compila (el intérprete lanza un error de sintaxis antes comenzar a ejecutarlo): No se puede invocar a la función *mensaje()* y asignarla en una variable, ya que esa función es de la forma sin retorno de valor.

c.
El programa funciona y hace exactamente lo esperado, pero la función *mensaje()* es una función sin retorno de valor, por lo que no debería ser asignada en una variable cuando se la invoca. ✓

¡Ok!

d.
El programa funciona, hace exactamente lo esperado, y no presenta ningún tipo de inconveniente ni elementos extraños en su código fuente.

¡Correcto!

La respuesta correcta es:

El programa funciona y hace exactamente lo esperado, pero la función *mensaje()* es una función sin retorno de valor, por lo que no debería ser asignada en una variable cuando se la invoca.

**Pregunta 7**

Correcta

Puntúa 1 sobre 1

Suponga que se quiere desarrollar un programa para cargar un número por teclado y mostrar el triple de ese número en consola de salida ¿Está bien planteado el siguiente programa?

```
__author__ = 'Cátedra de AED'


def triple(n):
    t = 3 * n


def test():
    a = int(input('Ingrese un numero: '))
    r = triple(a)
    print('El triple del numero es:', r)


# script principal
test()
```

Seleccione una:

a.
El programa lanza un error de intérprete: el parámetro formal de la función *triple()* no puede llamarse *n* (debería llamarse *a*).

b.
El programa ejecuta sin lanzar error, pero muestra un resultado *None* en lugar del triple del número cargado: falta la instrucción *return t* al final de la función *triple()*. ✓

¡Ok!

c.
El programa lanza un error de intérprete: la función *triple()* está definida como una función sin retorno de valor, y se producirá un error al invocarla y asignar su retorno en *t*.

d.
Sí, el programa hace exactamente lo esperado.

¡Correcto!

La respuesta correcta es:
El programa ejecuta sin lanzar error, pero muestra un resultado *None* en lugar del triple del número cargado: falta la instrucción *return t* al final de la función *triple()*.

**Pregunta 8**

Correcta

Puntúa 1 sobre 1

¿Qué hace la siguiente función en Python?

```
def check(n):
    if n % 2 == 0:
        return True
    else:
        return False
```

Seleccione una:

a.
Retorna True si el número n tomado como parámetro es primo.

b.
Retorna True si el número n tomado como parámetro es par. ✓

¡Ok!

c.
Retorna True si el número n tomado como parámetro es mayor a 2.

d.
Retorna True si el número n tomado como parámetro es impar.

¡Correcto!

La respuesta correcta es:
Retorna True si el número n tomado como parámetro es par.

**Pregunta 9**

Correcta

Puntúa 1 sobre 1

¿Qué efecto produce la ejecución del siguiente script, tal y como se muestra (exactamente con los valores 3 y 0 indicados como parámetro al invocar a la función *calcular()*)?

```
__author__ = 'Catedra de AED'


def calcular(a, b):
    if b != 0:
        c = a // b
        r = a % b
        return c, r

# script principal
a, b = calcular(3, 0)
print('Cociente:', a, 'Resto:', b)
```

Seleccione una:

a.
Ejecuta sin problemas. Muestra el mensaje "*Cociente: None  Resto: None*"

b.
Ejecuta sin problemas. Muestra el mensaje "*Cociente: 0  Resto: 0*"

c.
Se produce un error al intentar ejecutar la instrución `a, b = calcular(3, 0)`. ✓

¡Correcto! La función estaría retornando UN None (y no una tupla), con lo cual la asignación en dos variables para desempaquetar la tupla no es válida. El intérprete lanza un error.

d.
Ejecuta sin problemas. Muestra el mensaje "*None*"

¡Correcto!

La respuesta correcta es:
Se produce un error al intentar ejecutar la instrución `a, b = calcular(3, 0)`.

**Pregunta 10**

Correcta

Puntúa 1 sobre 1

El siguiente programa carga por teclado dos números, usa una función para obtener el mayor entre ambos, y finalmente muestra el mayor. ¿Está bien planteado el programa?

```
__author__ = 'Cátedra de AED'


def comprobar(n1, n2):
    if n1 > n2:
        return n1


def test():
    n1 = int(input('Ingrese un número: '))
    n2 = int(input('Ingrese otro: '))
    r = comprobar(n1, n2)
    print('El mayor es:', r)


# script principal
test()
```

Seleccione una:

a.

No. El programa ejecuta sin problemas pero siempre muestra como mayor al primer número, ya que no queda claro lo que pasa si el mayor fuese el segundo.

b.

No. El programa lanza un error de intérprete en la función comprobar(), ya que no tiene return en la rama falsa.

c.

Sí. Está correctamente planteado

d.

No. El programa muestra el valor None si el mayor es el segundo numero. La función comprobar() Debería tener un return en cada rama lógica que se plantee dentro de ella (falta un return en la rama falsa...)

¡Ok!

¡Correcto!

La respuesta correcta es:

No. El programa muestra el valor None si el mayor es el segundo numero. La función comprobar() Debería tener un return en cada rama lógica que se plantee dentro de ella (falta un return en la rama falsa...)

**Pregunta 11**

Correcta

Puntúa 1 sobre 1

El siguiente es un programa simple, que carga por teclado un número y usa una función para chequear si el mismo es cero o positivo (en cuyo caso, avisa con un mensaje) ¿Hay algún problema en el programa mostrado?

```
__author__ = 'Cátedra de AED'


def mostrar(n):
    if n < 0:
        return
    print('El número', n, 'es válido')


def test():
    n = int(input('Ingrese un número: '))
    mostrar(n)
    print('Programa terminado...')


# script principal
test()
```

Seleccione una:

a.
El programa ejecuta sin problemas, pero SIEMPRE muestra el mensaje que indica que el número es válido (debería mostrarlo sólo si el número es mayor o igual a cero).

b.
El programa lanza un error de intérprete: no se puede usar *return* en una función sin indicarle el valor a retornar.

c.
El programa lanza un error de intérprete: una función que usa un *return* sin valor indicado, no puede tomar parámetros.

d.
No hay ningún problema: el programa hace exactamente lo esperado y no tiene elementos extraños en su planteo. ✓

¡Ok!

¡Correcto!

La respuesta correcta es:
No hay ningún problema: el programa hace exactamente lo esperado y no tiene elementos extraños en su planteo.

**Pregunta 12**

Correcta

Puntúa 2 sobre 2

Considere el programa para el *Juego del Número Secreto* que se presentó en la Ficha 7. Ese programa se planteó originalmente sin funciones, y aquí lo mostramos redefinido para emplear funciones. Pero además, en la versión original del programa se usaba una *bandera* para marcar en el algoritmo si el número secreto fue encontrado o no; y nos proponemos ahora tratar de eliminar el uso de esa bandera y simplificar la estructura del programa. Sugerimos la modificación que se muestra más abajo empleando una *instrucción* ***return*** *para cortar el ciclo y la función apenas se encuentre el número secreto*. ¿Funciona correctamente el programa que estamos sugiriendo? *Seleccione la respuesta que mejor describa lo que está pasando con el programa propuesto.*

```
__author__ = 'Catedra de AED'

import random


def play_secret_number_game(limite_derecho, cantidad_inte
ntos):

    # limites iniciales del intervalo de búsqueda...
    izq, der = 1, limite_derecho

    # contador de intentos...
    intentos = 0

    # el numero secreto...
    secreto = random.randint(1, limite_derecho)

    # el ciclo principal... siga mientras no
    # haya sido encontrado el número, y la
    # cantidad de intentos no llegue a 5...
    while intentos < cantidad_intentos:
        intentos += 1
        print('\nEl numero está entre', izq, 'y', der)

        # un valor para forzar al ciclo a ser [1, N]...
        num = izq - 1

        # carga y validación del número sugerido por el u
suario...
        while num < izq or num > der:
            num = int(input('[Intento: ' + str(intentos)
 + '] => Ingrese su numero: '))
            if num < izq or num > der:
                print('Error... le dije entre', izq, 'y',
 der, '...')

        # controlar si num es correcto, avisar y cortar e
l ciclo...
        if num == secreto:
            print('\nGenio!!! Acertaste en', intentos, 'i
ntentos')
            return

        # ... pero si no lo es, ajustar los límites
        # del intervalo de búsqueda... y seguir...
        elif num > secreto:
            der = num
        else:
            izq = num

    print('\nLo siento!!! Se te acabaron los intentos. El
 número era:', secreto)
```

```
def test():
    print('Juego del Número Secreto... Configuración Inic
ial...')
    ld = int(input('El número secreto estará entre 1 y:
 '))
    ci = int(input('Cantidad máxima de intentos que tendr
á disponible: '))
    play_secret_number_game(ld, ci)


# script principal...
test()
```

Seleccione una:

a.
No. No funciona bien: en la forma en que está planteado, se muestran en forma incorrecta los mensajes informando los límites del intervalo que contiene al número secreto en cada vuelta del ciclo, ya que las variables *izq* y *der* se actualizan en forma incorrecta.

b.
Sí. Funciona correctamente para todos los casos. ✔

¡Correcto!

c.
No. No funciona bien: en la forma en que está planteado, cuando el jugador adivine el número secreto se mostrará el mensaje avisándole que ganó pero el ciclo continuará pidiendo que se ingrese un número, hasta agotar el número de intentos disponible.

d.
No. No funciona bien: en la forma en que está planteado, cuando el jugador adivine el número secreto la función *play_secret_number_game()* mostrará correctamente el mensaje avisando que ganó y cortará el ciclo con la instrucción *return*. Pero luego la función continuará e inmediatamente mostrará también el mensaje avisando que el jugador perdió, provocando ambigüedad.

**¡Correcto!**

La respuesta correcta es:
Sí. Funciona correctamente para todos los casos.

**Pregunta 13**

Correcta

Puntúa 2 sobre 2

El siguiente programa usa una función para verificar si un número que se carga por teclado es mayor que el triple de otro número que también se carga. ¿Cuál es el efecto de ejecutar esta función?

```
__author__ = 'Catedra de AED'


def ejemplo():
    i = int(input('Ingrese un valor: '))
    t = int(input('Ingrese otro: '))
    if i > 3*t:
        ok = True

    if ok:
        print('El primer valor es mayor al triple del seg
undo...')


# script principal
ejemplo()
```

Seleccione una:

a.
Mostrará el mensaje *El primer valor es mayor al triple del segundo* sólo si el valor cargado en *i* es menor al valor de *t* multiplicado por 3. No hará nada en caso contrarío.

b.
Mostrará el mensaje *El primer valor es mayor al triple del segundo* sólo si el valor cargado en *i* es menor o igual al valor cargado en *t* multiplicado por 3. No hará nada en caso contrario.

c.
Mostrará el mensaje *El primer valor es mayor al triple del segundo* sean cuales sean los valores que se carguen en *i* y en *t*.

d.
Mostrará el mensaje *El primer valor es mayor al triple del segundo* si *i > 3*t*, pero lanzará un error y se interrumpirá si *i <= 3*t*, ya que en este caso la variable *ok* quedaría sin definir. ✓

¡Ok!

¡Correcto!

La respuesta correcta es:
Mostrará el mensaje *El primer valor es mayor al triple del segundo* si *i > 3*t*, pero lanzará un error y se interrumpirá si *i <= 3*t*, ya que en este caso la variable *ok* quedaría sin definir.

**Pregunta 14**

Correcta

Puntúa 2 sobre 2

¿En cuáles de los siguientes casos el *algoritmo de divisiones sucesivas* es ***efectivamente muy lento e impráctico*** para determinar la primalidad de un número *n*? (En otras palabras: ¿qué caracterísitcas debe tener el número *n* para que el algoritmo caiga en un *peor caso* o cerca de un peor caso?) (Observación: más de una respuesta puede ser válida. Marque todas las que considere correctas)

Seleccione una o más de una:

- [ ] a.
Que *n* sea un número par.

- [ ] b.
Que *n* sea un número impar

- [x] c.
Que *n* sea un número compuesto (no primo) pero tal que el primero de sus divisores sea a su vez un número muy grande. ✓

**¡Correcto!** En este caso, el algoritmo deberá hacer ***muchas*** de todas las divisiones posibles... esto no cae necesariamente en el peor caso, pero se aproxima bastante...

- [x] d.
Que *n* sea ya un número primo muy grande (por ejemplo: $n > 10^{20}$) ✓

**¡Correcto!** En este caso, el algoritmo deberá hacer ***todas*** las divisiones posibles, que como vimos serán muchas... y caerá en el peor caso posible.

**¡Correcto!**

Las respuestas correctas son:
Que *n* sea ya un número primo muy grande (por ejemplo: $n > 10^{20}$),

Que *n* sea un número compuesto (no primo) pero tal que el primero de sus divisores sea a su vez un número muy grande.

**Pregunta 15**

Correcta

Puntúa 3 sobre 3

Se tiene un número entero positivo *n* de 40 dígitos ($n >= 10^{40}$) y se quiere determinar si *n* es primo aplicando el *algoritmo de divisiones sucesivas* visto en la sección de *Temas Avanzados* de la *Ficha 9* (básicamente, controlando todo divisor posible en el intervalo $[2..\sqrt{n}]$) Si nuestra computadora ejecuta unas mil millones de divisiones por segundo (1000000000 de divisiones por segundo = $10^9$ divisiones por segundo), ¿cuánto tiempo le llevaría en el peor caso a esa computadora determinar si *n* es primo? (Sugerencia: analice con cuidado la Ficha 9, página 219...)

Seleccione una:

a.
Alrededor de una hora.

b.
Alrededor de un día.

c.
Alrededor de una semana.

d.
Alrededor de un año.

e.
Alrededor de cien años.

f.
Alrededor de mil años.

g.
Alrededor de tres mil años. ✓

¡Correcto! En las condiciones dadas, el cálculo da aproximadamente 3170 años.

h.
Alrededor de treinta mil años.

i.
Alrededor de trescientos mil años.

j.
Alrededor de trescientos millones de años.

**¡Correcto!**

La respuesta correcta es:

Alrededor de tres mil años.

**Pregunta 16**

Correcta

Puntúa 2 sobre 2

¿Cuáles de las siguientes afirmaciones son ***verdaderas*** respecto de la *descomposición en factores primos* (o *factorización*) de un número *n entero y positivo*? (Observación: más de una respuesta puede ser válida. Marque todas las que considere correctas)

Seleccione una o más de una:

- [x] a.
  La factorización de todo número *n* entero y positivo es única. ✓

  **¡Correcto!** La demostración de este hecho constituye hoy el Teorema Fundamental de la Aritmética, comprobado ya por Euclides.

- [x] b.
  La factorización de un número *n* entero y positivo puede contener más de una vez al mismo factor primo. ✓

  **¡Correcto!** Basta con poner un ejemplo... si n = 24 la factorización es 2 * 2 * 2 * 3 con lo que el factor 2 aparece más de una vez...

- [x] c.
  Ningún número entero impar *n* > 2 puede contener al 2 en su factorización. ✓

  **¡Correcto!** Si *n* > 2 contiene al 2 como factor primo, entonces *n* es divisible por 2 y por lo tanto no sería impar...

- [ ] d.
  Ningún número entero impar *n* > 2 puede contener números impares en su factorización.

**¡Correcto!**

Las respuestas correctas son:
La factorización de todo número *n* entero y positivo es única.,

La factorización de un número *n* entero y positivo puede contener más de una vez al mismo factor primo.,

Ningún número entero impar *n* > 2 puede contener al 2 en su factorización.