Comenzado el	domingo, 18 de junio de 2023, 15:26
Estado	Finalizado
Finalizado en	domingo, 18 de junio de 2023, 15:59
Tiempo empleado	32 minutos 57 segundos
Puntos	15/16
Calificación	**10** de 10 (**96**%)

Pregunta **1**

Correcta

Se puntúa 1 sobre 1

En el contexto del diseño y planteo de funciones: ¿Qué diferencia existe entre *parámetros formales* y *parámetros actuales*?

Seleccione una:

a. Los *parámetros formales* son los que se envían a esa función al momento de invocarla, y los *parámetros actuales* son los que se declaran en la cabecera de la función al definir esa función.

b. Los *parámetros formales* son los que se declaran en la cabecera de la función al definir esa función, y los *parámetros actuales* son los que se envían a esa función al momento de invocarla. ✔

c. Los *parámetros formales* son los procesos que se declaran en el bloque de acciones de la función al definir esa función, y los *parámetros actuales* son los procesos dentro de los cuales se quiere usar a esa función al momento de invocarla.

d. Las designaciones de *parámetros formales* y *parámetros actuales* son equivalentes, y en ambos casos se refieren a cualquier variable o dato que se escriba entre los paréntesis de una función sin importar si se las está definiendo o se la está invocando.

Pregunta **2**

Correcta

Se 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: 0 Resto: 0*"

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

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

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

Pregunta **3**

Correcta

Se 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. Sí. Está correctamente planteado
- c. No. El programa lanza un error de intérprete en la función comprobar(), ya que no tiene return en la rama falsa.
- 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...) ✔

Pregunta **4**

Correcta

Se 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. No hay ningún problema: el programa hace exactamente lo esperado y no tiene elementos extraños en su planteo. ✔
- d. El programa lanza un error de intérprete: una función que usa un *return* sin valor indicado, no puede tomar parámetros.

Pregunta **5**

Correcta

Se 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_intentos):

    # 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 usuario...
        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 el ciclo...
        if num == secreto:
            print('\nGenio!!! Acertaste en', intentos, 'intentos')
            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 Inicial...')
    ld = int(input('El número secreto estará entre 1 y: '))
    ci = int(input('Cantidad máxima de intentos que tendrá 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. ✔

c. 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.

d. 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.

Pregunta **6**

Correcta

Se puntúa 1 sobre 1

En el contexto del diseño y planteo de funciones: ¿Qué diferencia existe entre una *variable local* y una *variable global*?

Seleccione una:

a. Una *variable global* se define dentro de una función y solo existe y puede usarse dentro de esa función, mientras que una *variable local* en general se define en el ámbito del script principal, fuera de cualquier función, y en principio es visible y puede usarse en cualquier lugar del programa (ya sea en el propio script principal o en cualquier función, salvo que la función tenga una variable global con el mismo nombre que la local).

b. Las designaciones de *variable local* y *variable global* son equivalentes, y en ambos casos se refieren a cualquier variable que se use dentro de una función o en el script principal.

c. Una *variable local* es exclusivamente un parámetro que se define entre los paréntesis de una función al definir a esa función (y en ese sentido, la expresión *variable local* es equivalente a la expresión *parámetro formal*), mientras que una *variable global* es un parámetro que se envía a una función entre sus paréntesis al momento de invocar a esa función (y en ese sentido, la expresión *variable global* es equivalente a la expresión *parámetro actual*).

d. Una *variable local* se define dentro de una función y solo existe y puede usarse dentro de esa función, mientras que una *variable global* en general se define en el ámbito del script principal, fuera de cualquier función, y en principio es visible y puede usarse en cualquier lugar del programa (ya sea en el propio script principal o en cualquier función, salvo que la función tenga una variable local con el mismo nombre que la global). ✔

Pregunta **7**

Correcta

Se 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 segundo...')


# script principal
ejemplo()
```

Seleccione una:

a. 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*.

b. 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.

c. 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.

d. 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.

Pregunta **8**

Correcta

Se 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 impar

b. Que *n* sea un número par.

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. ✔

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

Pregunta **9**

Correcta

Se 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..√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?

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. ✔
- h. Alrededor de treinta mil años.
- i. Alrededor de trescientos mil años.
- j. Alrededor de trescientos millones de años.

Pregunta **10**

Parcialmente correcta

Se puntúa 1 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:

- a. La factorización de todo número *n* entero y positivo es única. ✔
- b. La factorización de un número *n* entero y positivo puede contener más de una vez al mismo factor primo.
- c. Ningún número entero impar *n* > 2 puede contener al 2 en su factorización. ✔
- d. Ningún número entero impar *n* > 2 puede contener números impares en su factorización.

◄ Materiales Adicionales para la Ficha 11

Ir a...

Guía 11 de Ejercicios Prácticos ►