Comenzado el	domingo, 14 de mayo de 2023, 13:13
Estado	Finalizado
Finalizado en	domingo, 14 de mayo de 2023, 14:23
Tiempo empleado	1 hora 10 minutos
Puntos	16/16
Calificación	**10** de 10 (**100**%)

Pregunta **1**

Correcta

Se puntúa 1 sobre 1

¿Qué hace el siguiente programa, tal como está escrito, en Python?

```
__author__ = 'Catedra de AED'


c = 0
while c < 3:
    c += 1
print('Vuelta numero:', c)
```

Seleccione una:

a. Muestra un único mensaje: Vuelta número: 3 ✔ ¡Correcto! El ciclo en cada vuelta sólo ejecuta la instrucción c += 1, ya que su bloque de acciones sólo contiene a esa instrucción. La salida por pantalla está FUERA del bloque, ya que está indentada fuera de este, y se ejecuta sólo al terminar el ciclo, una única vez... ¿Comprende ahora la importancia de respetar la indentación al escribir un programa en Python?

b. Provoca un ciclo infinito.

c. No llega a mostrar nada: el ciclo corta sin entrar.

d. Muestra tres mensajes en líneas separadas, con vueltas numeradas del 1 al 3.

¡Correcto!

Pregunta **2**

Correcta

Se puntúa 1 sobre 1

¿Qué efecto provoca el siguiente programa?

```
__author__ = 'Catedra de AED'


c = 0
while c <= 100:
    print('Vuelta numero:', c)
    c -= 1

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

Seleccione una:

a. Un ciclo de 100 repeticiones con 100 mensajes, y al final el mensaje "*Programa Terminado*".

b. La aparición del mensaje "*Vuelta numero 0*" y luego el mensaje "*Programa Terminado*" (el ciclo hace sólo una repetición).

c. Un ciclo infinito. ✔ ¡Correcto! La variable c disminuye su valor en cada vuelta, por lo cual se aleja cada vez más del valor 101 que produciría el corte del ciclo.

d. La aparición del mensaje "*Programa Terminado*" (el ciclo no hace ninguna repetición).

¡Correcto!

Pregunta **3**

Correcta

Se puntúa 1 sobre 1

Supongamos que se desea mostrar en consola estándar los primeros *n* números impares. ¿Es correcto el siguiente script Python? (para simplificar, asuma que el usuario cargará un valor no negativo en ***n***):

```
c = 1
n = int(input('Cuántos impares quiere?: '))
while c <= 2*n:
    print(c)
c += 2
```

Seleccione una:

a. No. El script mostrado muestra números pares en lugar de impares (ya que el contador ***c*** avanza de a 2 en 2)

b. No. El script mostrado produce un ciclo infinito, ya que la instrucción ***c +=2*** está fuera del bloque del ciclo. ✔ ¡Correcto!

c. Sí. El script es correcto.

d. No. El script muestra ***2*n*** números impares, en lugar de sólo ***n***.

¡Correcto!

Pregunta **4**

Correcta

Se puntúa 2 sobre 2

Suponga que se pide cargar por teclado un conjunto de números, sin saber cuántos son. Se sabe que al cargar el número 0 se termina la carga de datos. Y se pide calcular el promedio entero de los números cargados. ¿Es correcto el planteo del siguiente programa? (Aclaración: el programa hace lo pedido... pero queremos saber si el planteo es óptimo, o está haciendo algo que no debería (aunque llegue de todos modos al resultado correcto) y/o podría mejorarse en alguna forma...)

```
__author__ = 'Catedra de AED'

c = a = p = 0

x = int(input('Ingrese un numero (con cero termina el proceso): '))
while x != 0:
    c += 1
    a += x
    p = a // c
    x = int(input('Ingrese otro (con cero termina el proceso): '))

print('El promedio es:', p)
```

Seleccione una:

a. El programa no calcula correctamente el promedio: la división está al revés.

b. El programa no sólo no calcula correctamente el promedio, sino que además se interrumpe en forma inesperada pues en la primera vuelta del ciclo divide por cero.

c. El programa calcula correctamente el promedio, y no hay necesidad de ninguna mejora.

d. El programa calcula el valor correcto del promedio, pero la división debería hacerse fuera del ciclo, al terminar el mismo. ✔

¡Correcto! Así como está hecho, sólo la última división realizada en la última vuelta dará el promedio final... por lo que no hay ninguna necesidad de perder tiempo dividiendo en cada vuelta los valores parciales del acumulador y el contador.

¡Correcto!

Pregunta **5**

Correcta

Se puntúa 3 sobre 3

Suponga que se le pide plantear un programa para cargar una secuencia de números hasta que aparezca el 0, y determinar si esa secuencia está ordenada de menor a mayor o no. Una forma correcta de hacerlo, se en el siguiente programa:

```
ok = True
num = int(input('Cargue el primer valor (con 0 corta): '))
anterior = num

while num != 0:
    if num < anterior:
        ok = False
    anterior = num
    num = int(input('Cargue el siguiente valor (con 0 corta): '))

if ok == True:
    print('La secuencia está ordenada...')
else:
    print('La secuencia no está ordenada')
```

Suponga que se modifica el modelo anterior y se reemplaza por el que se muestra más abajo. ¿Funciona correctamente el nuevo esquema?

```
ok = True
num = int(input('Cargue el primer valor (con 0 corta): '))
anterior = num

while num != 0:
    if num < anterior:
        ok = False
    else:
        ok = True
    anterior = num
    num = int(input('Cargue el siguiente valor (con 0 corta): '))

if ok == True:
    print('La secuencia está ordenada...')
else:
    print('La secuencia no está ordenada')
```

Seleccione una:

- a. Sí. Funciona correctamente con el cambio propuesto, aunque ya funcionaba también sin ese cambio. Por lo tanto, el cambio es redundante.
- b. No. No funciona correctamente. Con el cambio sugerido aquí, la función determina si la secuencia está ordenada pero de mayor a menor en lugar de hacerlo de menor a mayor.
- c. No. No funciona correctamente. Si secuencia fuese de la forma {2, 4, 6, 5, 7, 9} entonces al ingresar el 5 el flag *ok* ✔ ¡Ok! cambiará a *False (*correctamente) pero luego al ingresar el 7 volverá a cambiar a *True* (incorrectamente). Una vez que el *flag* cambia *False*, no debe volver a *True* durante el resto de la carga.
- d. No. No funciona correctamente. De hecho, el cambio sugerido provoca un error de intérprete y el programa lanza un error y se interrumpe si en la primera vuelta del ciclo entra por el else de esa condición.

¡Correcto!

Pregunta **6**

Correcta

Se puntúa 2 sobre 2

El siguiente es el programa original que se presentó en la Ficha 6 para resolver el problema número 15 (cargar por doble lectura una secuencia de datos hasta que aparezca el cero, y calcular cuántos de esos datos estaban en ciertos intervalos, calcular además la suma de todos los datos, y finalmente indicar si alguno de los datos era igual a cero):

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

# inicialización de contadores, acumuladores y banderas...
c1, c2, c3, t = 0, 0, 0, 0
ok = False

# proceso de carga por doble lectura...
# ... hacer la primera lectura...
cant = int(input('Ingrese cantidad vendida (con -1 termina el proceso): '))
while cant != -1:

    # punto a): chequear en qué intervalo está cada cantidad y contar...
    if 0 <= cant < 10000:
        c1 += 1
    elif 10000 <= cant < 15000:
        c2 += 1
    else:
        c3 += 1

    # punto b): acumular cada cantidad...
    t += cant

    # punto c): chequear si cant es 0, y marcar con un flag...
    if cant == 0:
        ok = True

    # hacer la segunda lectura...
    cant = int(input('Ingrese otra cantidad (con -1 termina): '))

# visualización de resultados... punto a)
print()
print('Cantidad de valores >= 0 pero < 10000:', c1)
print('Cantidad de valores >= 10000 pero < 15000:', c2)
print('Cantidad de valores >= 15000:', c3)

# visualización de resultados... punto b)
print('Cantidad total de vehículos vendidos:', t)

# visualización de resultados... punto c)
# recuerde que lo que sigue es equivalente a if ok == True:
if ok:
    print('Se registró al menos una cantidad de ventas igual cero')
else:
    print('No se registró ninguna cantidad de ventas igual cero')
```

Y el programa que sigue es una modificación del anterior, de forma que se ha eliminado la **segunda lectura** en la carga de datos. ¿Cuál es el efecto que tiene la eliminación de la **segunda lectura** en el programa modificado?

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

# inicialización de contadores, acumuladores y banderas...
c1, c2, c3, t = 0, 0, 0, 0
ok = False

# proceso de carga por doble lectura...
# ... hacer la primera lectura...
cant = int(input('Ingrese cantidad vendida (con -1 termina el proceso): '))
while cant != -1:

    # punto a): chequear en qué intervalo está cada cantidad y contar...
    if 0 <= cant < 10000:
        c1 += 1
    elif 10000 <= cant < 15000:
        c2 += 1
    else:
        c3 += 1

    # punto b): acumular cada cantidad...
    t += cant

    # punto c): chequear si cant es 0, y marcar con un flag...
    if cant == 0:
        ok = True

# visualización de resultados... punto a)
print()
print('Cantidad de valores >= 0 pero < 10000:', c1)
print('Cantidad de valores >= 10000 pero < 15000:', c2)
print('Cantidad de valores >= 15000:', c3)

# visualización de resultados... punto b)
print('Cantidad total de vehículos vendidos:', t)

# visualización de resultados... punto c)
# recuerde que lo que sigue es equivalente a if ok == True:
if ok:
    print('Se registró al menos una cantidad de ventas igual cero')
else:
    print('No se registró ninguna cantidad de ventas igual cero')
```

Seleccione una:

- a. Si el primer dato que se cargue al ejecutarlo es el valor -1, el programa se detendrá normalmente y mostrará todos los resultados con sus valores iniciales. Pero si primer dato que se cargue no es -1, entonces el programa pedirá solamente la carga de un nuevo valor, y luego se detendrá mostrando los resultados de acuerdo a los dos únicos datos que aceptó.
- b. No produce ningún efecto especial. El nuevo programa funciona tan bien como el original.
- c. Si el primer dato que se cargue al ejecutarlo es el valor -1, el programa se detendrá normalmente y mostrará todos los resultados con sus valores iniciales. Pero si primer dato que se cargue no es -1, entonces el programa entrará en ciclo infinito. ✔

  ¡Correcto! Así como está hecho, al terminar de cargar el primer número (si este no es -1) parecerá que el programa no está haciendo nada (aunque de hecho estará girando una y otra vez con los mismos datos), pero tampoco pedirá ningún otro número.
- d. Si el primer dato que se cargue al ejecutarlo es el valor -1, el programa entrará en ciclo infinito. Pero si primer dato que se cargue no es -1, entonces el programa funcionará correctamente

¡Correcto!

Pregunta **7**

Correcta

Se puntúa 2 sobre 2

El siguiente es ***OTRA VEZ*** el programa original que se presentó en la Ficha 6 para resolver el problema número 15 (cargar por doble lectura una secuencia de datos hasta que aparezca el cero, y calcular cuántos de esos datos estaban en ciertos intervalos, calcular además la suma de todos los datos, y finalmente indicar si alguno de los datos era igual a cero):

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

# inicialización de contadores, acumuladores y banderas...
c1, c2, c3, t = 0, 0, 0, 0
ok = False

# proceso de carga por doble lectura...
# ... hacer la primera lectura...
cant = int(input('Ingrese cantidad vendida (con -1 termina el proceso): '))
while cant != -1:

    # punto a): chequear en qué intervalo está cada cantidad y contar...
    if 0 <= cant < 10000:
        c1 += 1
    elif 10000 <= cant < 15000:
        c2 += 1
    else:
        c3 += 1

    # punto b): acumular cada cantidad...
    t += cant

    # punto c): chequear si cant es 0, y marcar con un flag...
    if cant == 0:
        ok = True

    # hacer la segunda lectura...
    cant = int(input('Ingrese otra cantidad (con -1 termina): '))

# visualización de resultados... punto a)
print()
print('Cantidad de valores >= 0 pero < 10000:', c1)
print('Cantidad de valores >= 10000 pero < 15000:', c2)
print('Cantidad de valores >= 15000:', c3)

# visualización de resultados... punto b)
print('Cantidad total de vehículos vendidos:', t)

# visualización de resultados... punto c)
# recuerde que lo que sigue es equivalente a if ok == True:
if ok:
    print('Se registró al menos una cantidad de ventas igual cero')
else:
    print('No se registró ninguna cantidad de ventas igual cero')
```

Y el programa que sigue es una modificación del anterior, de forma que se ha eliminado la **primera lectura** en la carga de datos. ¿Cuál es el efecto que tiene en Python la eliminación de la **primera lectura** en el programa modificado dejándolo tal cual se ve a continuación?

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

# inicialización de contadores, acumuladores y banderas...
c1, c2, c3, t = 0, 0, 0, 0
ok = False

# proceso de carga por doble lectura...

while cant != -1:

    # punto a): chequear en qué intervalo está cada cantidad y contar...
    if 0 <= cant < 10000:
        c1 += 1
    elif 10000 <= cant < 15000:
        c2 += 1
    else:
        c3 += 1

    # punto b): acumular cada cantidad...
    t += cant

    # punto c): chequear si cant es 0, y marcar con un flag...
    if cant == 0:
        ok = True

    # hacer la segunda lectura...
    cant = int(input('Ingrese otra cantidad (con -1 termina): '))

# visualización de resultados... punto a)
print()
print('Cantidad de valores >= 0 pero < 10000:', c1)
print('Cantidad de valores >= 10000 pero < 15000:', c2)
print('Cantidad de valores >= 15000:', c3)

# visualización de resultados... punto b)
print('Cantidad total de vehículos vendidos:', t)

# visualización de resultados... punto c)
# recuerde que lo que sigue es equivalente a if ok == True:
if ok:
 print('Se registró al menos una cantidad de ventas igual cero')
else:
 print('No se registró ninguna cantidad de ventas igual cero')
```

Seleccione una:

- a. Si el primer dato que se cargue al ejecutarlo es el valor -1, el programa entrará en ciclo infinito. Pero si primer dato que se cargue no es -1, entonces el programa funcionará correctamente
- b. No produce ningún efecto especial. El nuevo programa funciona tan bien como el original.
- c. Si el primer dato que se cargue al ejecutarlo es el valor -1, el programa se detendrá normalmente y mostrará todos los resultados con sus valores iniciales. Pero si primer dato que se cargue no es -1, entonces el programa pedirá solamente la carga de un nuevo valor, y luego se detendrá mostrando los resultados de acuerdo a los dos únicos datos que aceptó.
- d. Al intentar ejecutar el programa modificado, se producirá un error de intérprete al hacer la comprobación del ciclo while, ya que así como está planteado la variable *cant* quedará indefinida al hacer la primera comprobación. ✔

¡Correcto! Así como está hecho, la variable *cant* nunca llega a ser definida antes de usarse en la primera comprobación del ciclo...

¡Correcto!

Pregunta **8**

Correcta

Se puntúa 1 sobre 1

Analice el siguiente script básico en Python:

```
x = -9
y = 2 + x**0.5
print(y)
```

¿Cuál es el efecto de ejecutar el script anterior?

Seleccione una:

a. Se ejecutará sin interrumpirse ni lanzar error, y la variable ***y*** quedará asignada con el valor *(2+3j)* de tipo *complex*. ✔ ¡Ok!

b. Se ejecutará sin interrumpirse ni lanzar error, pero la variable ***y*** quedará asignada con el valor *None* debido a que no puede calcular la raíz cuadrada de un número negativo.

c. Se ejecutará sin interrumpirse ni lanzar error, y la variable ***y*** quedará asignada con el valor *5.0* de tipo *float*.

d. Al ejecutarse se interrumpirá en forma abrupta, lanzando un mensaje de operación matemática incorrecta (raíz cuadrada de un número negativo).

¡Correcto!

Pregunta **9**

Correcta

Se puntúa 2 sobre 2

Analice el siguiente script básico en Python:

```
c1 = complex(3, 2)
c2 = complex('1 + 5j')
print('c1:', c1)
print('c2:', c2)
```

¿Cuál es el efecto de ejecutar el script anterior?

Seleccione una:

a. Ejecuta sin problemas, y muestra en consola de salida los complejos *c1: (5j)* y *c2: (6j)*.

b. Al ejecutarse se interrumpirá en forma abrupta, lanzando un mensaje de cadena mal formada en la asignacion *c2 = complex('1 + 5j')* (la cadena contiene espacios en blanco a los lados del signo '+'). ✔ ¡Ok!

c. Ejecuta sin problemas, y muestra en consola de salida los complejos *c1: (3+2j)* y *c2: (1+5j)*.

d. Al ejecutarse se interrumpirá en forma abrupta, lanzando un mensaje de operación imposible en la primera asignación *c1 = complex(3, 2)* (falta el factor *j* acompañando al 2).

¡Correcto!

Pregunta **10**

Correcta

Se puntúa 1 sobre 1

El siguiente es el programa publicado en la Ficha 6 que calcula las raíces de la ecuación de segundo grado en forma directa, dejando que Python se encargue de los detalles de manejo de los números complejos si fuese el caso:

```
__author__ = 'Cátedra de AED'

seguir = 's'
while seguir == 's' or seguir == 'S':

    # titulo y carga de datos...
    print('Raíces de la ecuación de segundo grado...')
    a = float(input('a: '))
    b = float(input('b: '))
    c = float(input('c: '))

    # procesos: aplicar directamente las fórmulas...
    x1 = (-b + (b**2 - 4*a*c)**0.5) / (2*a)
    x2 = (-b - (b**2 - 4*a*c)**0.5) / (2*a)

    # visualización de resultaddos...
    print('x1:', x1)
    print('x2:', x2)

    seguir = input('Desea resolver otra ecuacion? (s/n): ')

print('Gracias por utilizar nuestro programa...')
```

La consigna en esta pregunta es que se tome un tiempo para ejecutar y probar este programa, entender lo que ocurre, y simplemente registrar los resultados más abajo. En la columna de la izquierda figuran distintas ecuaciones de segundo grado, y en la columna de la derecha figuran las posibles raíces de cada una, en forma desordenana. Seleccione la solución correcta para cada ecuación (Nota: los decimales a la derecha del punto pueden aparecer redondeados o truncados. Seleccione la respuesta más aproximada si ese fuese el caso).

$x^2 + x + 1 = 0$	x1= (-0.5+0.86602j) x2= (-0.5-0.86602j) ✔
$2x^2 + 4x + 2 = 0$	x1= -1.0 x2= -1.0 ✔
$2x^2 = 0$	x1= 0 x2= 0 ✔
$3x^2 + 2x + 6 = 0$	x1 = (-0.33333+1.37436j) x2 = (-0.33333-1.37436j) ✔
$x^2 - 2x - 3 = 0$	x1= 3.0 x2= -1.0 ✔
$-4x^2 + 3x + 1 = 0$	x1= -0.25 x2= 1.0 ✔

¡Ok!

◄ Ficha Opcional 06 [Modelo de Aplicación Simple] (PDF - para lectura directa)

Ir a...

Guía de Ejercicios Prácticos - Ficha 06 ►