

Comenzado el	sábado, 9 de septiembre de 2023, 20:21
Estado	Finalizado
Finalizado en	sábado, 9 de septiembre de 2023, 20:42
Tiempo empleado	21 minutos 41 segundos
Puntos	17/17
Calificación	10 de 10 (98%)

Pregunta 1

Parcialmente correcta

Se puntúa 1 sobre 1

¿Cuáles de las siguientes son características básicas y generales de un *registro*? (Más de una respuesta puede ser válida. Marque todas las que considere correctas).

Seleccione una o más de una:

- ☐ a. Un registro es un conjunto inmutable de datos que pueden ser de tipos diferentes.
- ☒ b. Los elementos individuales de un registro se acceden por medio de un *nombre* o *identificador* declarado por el programador, en lugar de hacerlo por medio de índices. ✓
- ☒ c. Los elementos individuales de un registro se designan con el nombre genérico de *campos*. ✓
- ☐ d. Un registro es un conjunto mutable de datos que pueden ser de tipos diferentes.

Pregunta 2

Correcta

Se puntúa 1 sobre 1

Suponga que se quiere almacenar en un programa los datos de un *estudiante*, y para representar a ese estudiante se usa una *tupla* en la que el primer casillero guardará el legajo, el segundo el nombre y el tercero el promedio del estudiante. Suponga que parte del programa incluye una secuencia de instrucciones como la siguiente, en la cual se crea la tupla, se muestra su contenido, se cambia el nombre del estudiante y luego se muestran los datos modificados:

```
est = 12345, 'Juan', 8.76
print('Datos del estudiante:', est)

est[1] = 'Pedro'
print('Datos modificados del estudiante:', est)
```

¿Es correcto este enfoque o existe algún inconveniente? Si hay algún problema, indique cuál.

Seleccione una:

- ☐ a. Sí. Es correcto y el esquema mostrado funciona sin problemas.
- ☐ b. No es correcto. Los elementos de una tupla no pueden accederse usando índices.
- ☒ c. No es correcto. Una tupla podría usarse para representar al estudiante, pero no permitirá modificar ningún componente al ser una secuencia de tipo inmutable. ✓
- ☐ d. No es correcto. No se pueden almacenar valores de tipos diferentes en una tupla.

Pregunta **3**

Correcta

Se puntúa 1 sobre 1

Suponga que se quiere representar una fecha en un programa, y se propone utilizar un esquema basado en el uso de un registro, como se muestra en el siguiente script:

```
class Fecha:
    pass

f = Fecha()
f.day = 27
f.month = 7
f.year = 2015

print('Fecha registrada: ', f.day, '/', f.month, '/', f.year, sep='')
```

¿Es correcto este enfoque o existe algún inconveniente? Si hay algún problema, indique cuál.

Seleccione una:

- ☒ a. Sí. Es correcto y el esquema mostrado funciona sin problemas. ✓
- ☐ b. No es correcto. Un registro es un conjunto de datos que deben ser de tipos diferentes, y en este esquema todos los campos del registro *f* son del mismo tipo: *int*.
- ☐ c. No es correcto. La declaración de la clase *Fecha* no es válida: los campos deben declararse obligatoriamente dentro de ella (no puede dejarse vacía usando la instrucción *pass*).
- ☐ d. No es correcto. Los elementos de un registro no pueden accederse usando el operador *punto*.

Pregunta **4**

Correcta

Se puntúa 1 sobre 1

Suponga que se quiere representar datos de tres libros en un programa, y se propone utilizar un esquema basado en el uso de registros, como se muestra en el siguiente script:

```
class Libro:
    pass

lib1 = Libro()
lib1.codigo = 2345
lib1.titulo = 'El Aleph'

lib2 = Libro()
lib2.codigo = 1267
lib2.titulo = 'Rayuela'
lib2.autor = 'Julio Cortázar'

lib3 = Libro()
lib3.isbn = 123456767
lib3.nombre = 'El Tunel'
lib3.precio = 145.56
```

¿Es correcto el script mostrado o existe algún inconveniente? Si hay algún problema, indique cuál.

Seleccione una:

- ☐ a. No es correcto. Las tres variables *lib1*, *lib2* y *lib3* se están definiendo con *distinta cantidad de campos* y eso no es posible.
- ☐ b. No es correcto. Las tres variables *lib1*, *lib2* y *lib3* pueden tener campos con nombres diferentes o incluso distinta cantidad de campos, *pero al menos uno de los campos debe ser el mismo tipo para todos los registros*.
- ☒ c. Sí. Es correcto y el esquema mostrado funciona sin problemas. ✓
- ☐ d. No es correcto. Las tres variables *lib1*, *lib2* y *lib3* se están definiendo con *campos de nombres o identificadores diferentes* y eso no es posible.

Pregunta 5

Correcta

Se puntúa 1 sobre 1

Suponga que se quiere representar datos de distintos libros en un programa, y se propone utilizar un esquema basado en el uso de registros, como se muestra en el script de más abajo. ¿Hay algún problema con el programa siguiente?

```
class Libro:
    pass

def init(cod, nom, aut):
    libro = Libro()
    libro.codigo = cod
    libro.titulo = nom
    libro.autor = aut
    return libro

def write(libro):
    print('Datos del libro:')
    print('Código:', libro.codigo, ' - Título:', libro.titulo, ' - Autor:', libro.autor)

def test():
    lib1 = init(2345, 'El Aleph', 'Jorge Luis Borges')
    lib2 = init(1267, 'Rayuela', 'Julio Cortázar')
    lib3 = init(1928, 'El Túnel', 'Ernesto Sábato')

    write(lib1)
    write(lib2)
    write(lib3)

if __name__ == '__main__':
    test()
```

Seleccione una:

- ☐ a. No es correcto. Falta la creación del registro vacío en la función *test()* antes de invocar a *init()*.
- ☒ b. Sí. Es correcto y el esquema mostrado funciona sin problemas. ✓
- ☐ c. No es correcto. La función *init()* está creando nuevamente un registro vacío en su bloque de acciones (al hacer *libro = Libro()*). Cuando la función termina de ejecutarse, ese registro se pierde y el parámetro actual enviado a ella seguirá apuntando al registro vacío creado en *test()*. Cuando se invoque a la función *write()* se producirá entonces un error de intérprete porque intentará mostrar campos que no existen.
- ☐ d. No es correcto. La función *init()* no puede invocar a la función constructora (sólo puede invocarse en la función *test()* o en cualquier función que sea usada como función de arranque del programa).

Pregunta 6

Correcta

Se puntúa 1 sobre 1

Suponga que se quiere representar datos de distintos libros en un programa, y se propone utilizar un esquema basado en el uso de registros, como se muestra en el script de más abajo. ¿Hay algún problema con el programa siguiente?

```
class Libro:
    pass

def init(libro, cod, nom, aut):
    libro = Libro()
    libro.codigo = cod
    libro.titulo = nom
    libro.autor = aut

def write(libro):
    print('Datos del libro:')
    print('Código:', libro.codigo, ' - Título:', libro.titulo, ' - Autor:', libro.autor)

def test():
    lib1 = Libro()
    init(lib1, 2345, 'El Aleph', 'Jorge Luis Borges')


    lib2 = Libro()
    init(lib2, 1267, 'Rayuela', 'Julio Cortázar')

    lib3 = Libro()
    init(lib3, 1928, 'El Túnel', 'Ernesto Sábato')

    write(lib1)
    write(lib2)
    write(lib3)

if __name__ == '__main__':
    test()
```

Seleccione una:

- ☐ a. No es correcto. Si la función *init()* está creando el registro (al hacer *libro = Libro()*) entonces no debe crearse el registro también en *test()*. En *test()* es suficiente con invocar a *init()* para crear cada registro.
- ☒ b. No es correcto. La función *init()* está creando nuevamente un registro vacío en su bloque de acciones (al hacer *libro = Libro()*).  Cuando la función termina de ejecutarse, ese registro se pierde y el parámetro actual enviado a ella seguirá apuntando al registro vacío creado en *test()*. Cuando se invoque a la función *write()* se producirá entonces un error de intérprete porque intentará mostrar campos que no existen.
- ☐ c. No es correcto. La función *init()* no puede invocar a la función constructora (sólo puede invocarse en la función *test()* o en cualquier función que sea usada como función de arranque del programa).
- ☐ d. Sí. Es correcto y el esquema mostrado funciona sin problemas.

Pregunta 7

Correcta

Se puntúa 2 sobre 2

Suponga que se quiere representar datos de distintos libros en un programa, y se propone utilizar un esquema basado en el uso de registros, como se muestra en el script de más abajo. Suponga además que una vez creadas las variables *lib1*, *lib2* y *lib3* se quiere crear una nueva variable *lib4* que contenga los mismos datos que *lib2*, pero de tal forma que ambas se mantengan independientes: al modificar cualquier dato en una de las dos, no debe cambiar nada en la otra. ¿Hay algún problema con el programa siguiente, en base a estos requerimientos?

```
class Libro:
    pass

def init(libro, cod, nom, aut):
    libro.codigo = cod
    libro.titulo = nom
    libro.autor = aut

def write(libro):
    print('Datos del libro:')
    print('Código:', libro.codigo, ' - Título:', libro.titulo, ' - Autor:', libro.autor)

def test():

    lib1 = Libro()
    init(lib1, 2345, 'El Aleph', 'Jorge Luis Borges')

    lib2 = Libro()
    init(lib2, 1267, 'Rayuela', 'Julio Cortázar')

    lib3 = Libro()
    init(lib3, 1928, 'El Túnel', 'Ernesto Sábato')


    write(lib1)
    write(lib2)
    write(lib3)

    lib4 = lib2
    lib4.autor = 'Adolfo Bioy Casares'

    write(lib2)
    write(lib4)

if __name__ == '__main__':
    test()
```

Seleccione una:

- ☐ a. Sí. Es correcto y el esquema mostrado funciona sin problemas.
- ☒ b. No es correcto. La asignación *lib4 = lib2* hace que ambas variables queden referenciando al mismo registro. Cualquier cambio  que se haga de allí en más en una de las dos, cambiará entonces también a la otra.
- ☐ c. No es correcto. Antes de la asignación *lib4 = lib2* debería haberse creado la variable *lib4* con la función constructora *Libro()* (es decir: *lib4 = Libro()*) y sólo entonces asignar *lib2*.
- ☐ d. No es correcto. En Python no hay forma de hacer que dos variables de tipo registro contengan los mismos datos y se mantengan independientes la una de la otra.

Pregunta 8

Correcta

Se puntúa 2 sobre 2

(Esta pregunta contiene una modificación con respecto a la pregunta anterior... Revise bien el código fuente de ambas). Suponga que se quiere representar datos de distintos libros en un programa, y se propone utilizar un esquema basado en el uso de registros, como se muestra en el script de más abajo. Suponga además que una vez creadas las variables *lib1*, *lib2* y *lib3* se quiere crear una nueva variable *lib4* que contenga los mismos datos que *lib2*, pero de tal forma que ambas se mantengan independientes: al modificar cualquier dato en una de las dos, no debe cambiar nada en la otra. ¿Hay algún problema con el programa siguiente, en base a estos requerimientos?

```
class Libro:
    pass

def init(libro, cod, nom, aut):
    libro.codigo = cod
    libro.titulo = nom
    libro.autor = aut

def write(libro):
    print('Datos del libro:')
    print('Código:', libro.codigo, ' - Título:', libro.titulo, ' - Autor:', libro.autor)

def test():

    lib1 = Libro()
    init(lib1, 2345, 'El Aleph', 'Jorge Luis Borges')

    lib2 = Libro()
    init(lib2, 1267, 'Rayuela', 'Julio Cortázar')

    lib3 = Libro()
    init(lib3, 1928, 'El Túnel', 'Ernesto Sábato')

    write(lib1)
    write(lib2)
    write(lib3)

    lib4 = Libro()
    init(lib4, lib2.codigo, lib2.titulo, lib2.autor)
    lib4.autor = 'Adolfo Bioy Casares'

    write(lib2)
    write(lib4)

if __name__ == '__main__':
    test()
```

Seleccione una:

- ☒ a. Sí. Es correcto y el esquema mostrado funciona sin problemas. ✓
- ☐ b. No es correcto. La invocación a *init()* para inicializar *lib4* (*init(lib4, lib2.codigo, lib2.titulo, lib2.autor)*) hará que *lib2* y *lib4* apunten al mismo registro y desde allí en adelante cualquier cambio en una de ellas cambiará también la otra
- ☐ c. No es correcto. En Python no hay forma de hacer que dos variables de tipo registro contengan los mismos datos y se mantengan independientes la una de la otra.
- ☐ d. No es correcto. En lugar de invocar a *init()* para inicializar *lib4* con los datos de *lib2*, debería haberse asignado directamente *lib2* en *lib4* (o sea: *lib4 = Libro()* y luego *lib4 = lib2*).

Pregunta **9**

Correcta

Se puntúa 1 sobre 1

¿Qué diferencia básica existe entre un tipo de registro (entendido en el contexto de la clásico de Programación Estructurada (o PE)) y una clase (entendida en el contexto de la Programación Orientada a Objetos (o POO))?

Seleccione una:

- ☐ a. Una *clase* solo contiene datos (o campos) que describen la forma o el contenido de cada registro de ese tipo, mientras que un *tipo de registro* contiene tanto datos (o atributos) como procesos (o métodos) que describen respectivamente la forma (o contenido) y el comportamiento de cada objeto de esa clase.
- ☐ b. Un *tipo de registro* es una colección inmutable de datos, mientras que una *clase* es una colección mutable. En todo lo demás son equivalentes.
- ☒ c. Un *tipo de registro* solo contiene datos (o campos) que describen la forma o el contenido de cada registro de ese tipo, mientras que una *clase* contiene tanto datos (o atributos) como procesos (o métodos) que describen respectivamente la forma (o contenido) y el comportamiento de cada objeto de esa clase. ✓
- ☐ d. No hay ninguna diferencia. Son solo dos nombres diferentes para el mismo concepto.

Pregunta **10**

Correcta

Se puntúa 1 sobre 1

Todos los paradigmas de programación están basados en ciertas características y propiedades que los definen y los distinguen del resto de los paradigmas. ¿Cuáles son las propiedades o características esenciales del paradigma de la Programación Orientada a Objetos (o POO)?

Seleccione una:

- ☐ a. El *Encapsulamiento*, la *División en Subproblemas* y la *Modularización*.
- ☐ b. El *Garbage Collector*, la *Memoria Adaptativa* y la *Gestión de Excepciones*.
- ☐ c. La *Normalización*, la *Recursividad* y el *Diseño de Interfaces de Usuario*.
- ☒ d. El *Encapsulamiento*, la *Herencia* y el *Polimorfismo*. ✓

Pregunta **11**

Correcta

Se puntúa 1 sobre 1

¿Cuáles de las siguientes son ciertas en relación al *parámetro self* que todos los métodos de una clase en Python deben recibir? (Más de una respuesta puede ser correcta, por lo que marque todas las que considere válidas).

Seleccione una o más de una:

- ☒ a. El *parámetro self* debe ser definido usualmente en el primer lugar de la lista de parámetros formales de los métodos de una clase, pero luego al invocar al método no debe ser enviado en forma explícita como parámetro al método: Python asume que el objeto con el que se está llamando al método es el que se debe asignar en *self*. ✓
- ☐ b. El *parámetro self* es un flag que Python usa para indicar que la función que lo contiene es efectivamente un método de una clase.
- ☐ c. El *parámetro self* debe ser definido usualmente en el primer lugar de la lista de parámetros formales de los métodos de una clase, y luego al invocar al método debe ser enviado en forma explícita como primer parámetro el objeto sobre el cual el método debe aplicarse.
- ☒ d. El *parámetro self* es una referencia al objeto a través del cual se invocó al método (y ese es el objeto sobre el cual se aplicarán los procesos de ese método). ✓

Pregunta **12**

Correcta

Se puntúa 1 sobre 1

El siguiente módulo contiene la declaración de una clase *Estudiante*, incluyendo el constructor `__init__()` para crear e inicializar un objeto y el método `__str__()` para convertir a cadena un objeto. Analice la forma en que está planteado el método `__str__()`... ¿Qué efecto provocarán en la cadena retornada por ese método los *campos de reemplazo* usados al invocar al método `format()`?

```
class Estudiante:
    def __init__(self, leg, nom, prom):
        self.legajo = leg
        self.nombre = nom
        self.promedio = prom

    def __str__(self):
        r = ''
        r += '{:<15}'.format('Legajo: ' + str(self.legajo))
        r += '{:^30}'.format('Nombre: ' + self.nombre)
        r += '{:>18}'.format('Promedio: ' + str(self.promedio))
        return r
```

Seleccione una:

- ☐ a. El valor del atributo *legajo* sale centrado, el *nombre* también sale centrado y el *promedio* se justifica a la izquierda.
- ☐ b. El valor del atributo *legajo* se justifica a la izquierda, el *nombre* sale centrado y el *promedio* se justifica también a la izquierda.
- ☐ c. El valor del atributo *legajo* se justifica a la derecha, el *nombre* sale centrado y el *promedio* se justifica a la izquierda.
- ☒ d. El valor del atributo *legajo* se justifica a la izquierda, el *nombre* sale centrado y el *promedio* se justifica a la derecha. ✓

Pregunta **13**

Correcta

Se puntúa 1 sobre 1

¿Cuál es la diferencia *tipos de datos nativos* y *tipos de datos abstractos*?

Seleccione una:

- ☐ a. Los tipos abstractos vienen ya definidos y listos para usar en el lenguaje que se esté usando, mientras que los tipos nativos no están predefinidos y deben ser implementados por el programador.
- ☐ b. No hay diferencia alguna. Son dos formas de referirse al conjunto de tipos de datos disponibles en un lenguaje.
- ☒ c. Los tipos nativos vienen ya definidos y listos para usar en el lenguaje que se esté usando, mientras que los tipos abstractos no están predefinidos y deben ser implementados por el programador. ✓
- ☐ d. Los tipos nativos hacen referencia a estructuras inmutables de datos, mientras que los tipos abstractos refieren a colecciones mutables.

La clase Point analizada en clases tiene la siguiente estructura (considerando solo el constructor y el método gradient()):

```
class Point:
    def __init__(self, cx, cy, desc='p'):
        self.x = cx
        self.y = cy
        self.descripcion = desc

    def gradient(self, p2):
        dy = p2.y - self.y
        dx = p2.x - self.x

        if dx != 0:
            return dy / dx

        return None
```

Suponga ahora el siguiente script de prueba:

```
p = Point(2, 5)
q = Point(4, 6)
pd = p.gradient(q)
```

¿Cuáles de las siguientes afirmaciones son ciertas en relación a la invocación al método gradient() que se ve en ese script? (más de una puede ser cierta, por lo que marque todas las que considere correctas).

Seleccione una o más de una:

- ☐ a. Al invocar a *gradient()* en la forma indicada, se interrumpirá el programa lanzando un error de intérprete debido a que *gradient()* solicita dos parámetros pero solo se está enviando uno.
- ☐ b. Al invocar a *gradient()* en la forma indicada, el punto *q* será asignado en *self*, y el punto *p* será asignado en *p2*.
- ☒ c. Al invocar a *gradient()* en la forma indicada, el punto *p* será asignado en *self*, y el punto *q* será asignado en *p2*. ✓
- ☐ d. Al invocar a *gradient()* en la forma indicada, se producirá un error de intérprete por invocación incorrecta. En lugar de *p.gradient(q)* debería decir *gradient(p, q)*.

◀ [Materiales Adicionales para la Ficha 19](#)

Ir a...



[Desafío 03 \[Problema: Conteo de Frecuencias y Valor Modal\]](#) ▶