

Comenzado el	domingo, 20 de agosto de 2023, 15:48
Estado	Finalizado
Finalizado en	domingo, 20 de agosto de 2023, 17:17
Tiempo empleado	1 hora 29 minutos
Puntos	24/24
Calificación	10 de 10 (100%)

Pregunta **1**

Correcta

Se puntúa 2 sobre 2

El cálculo del valor a^n (para simplificar, asumimos $a > 0$ y $n \geq 0$ y sabiendo que si $n = 0$ entonces $a^0 = 1$) es igual a multiplicar n veces el número a por sí mismo. Por caso, $5^3 = 5 * 5 * 5 = 125$. Sabiendo esto, ¿cuál de las siguientes sería una *definición recursiva* matemáticamente correcta de la operación $potencia(a, n)$?

Seleccione una:

☐ a.

$$potencia(a, n) \begin{cases} = 1 & \text{si } n == 0 \\ = a * potencia(a, n) & \text{si } n > 0 \end{cases}$$

(a y n enteros, a > 0 y n ≥ 0)

☐ b.

$$potencia(a, n) \begin{cases} = 1 & \text{si } n == 0 \\ = a + potencia(a, n-1) & \text{si } n > 0 \end{cases}$$

(a y n enteros, a > 0 y n ≥ 0)

☒ c.

$$potencia(a, n) \begin{cases} = 1 & \text{si } n == 0 \\ = a * potencia(a, n-1) & \text{si } n > 0 \end{cases}$$

(a y n enteros, a > 0 y n ≥ 0)



☐ d.

$$potencia(a, n) \begin{cases} = 1 & \text{si } n == 0 \\ = a * potencia(a-1, n) & \text{si } n > 0 \end{cases}$$

(a y n enteros, a > 0 y n ≥ 0)

Pregunta **2**

Correcta

Se puntúa 1 sobre 1

Suponga que se quiere plantear una **definición recursiva** del concepto de **bosque**. ¿Cuál de las siguientes propuestas generales es correcta y constituye la mejor definición?

Seleccione una:

- ☐ a. Un bosque es un conjunto que puede contener uno o más árboles agrupados con otro bosque.
- ☒ b. Un bosque es un conjunto de árboles que puede estar vacío, o puede contener uno o más árboles agrupados con otro bosque. ✓
- ☐ c. Un bosque es un bosque.
- ☐ d. Un bosque es un conjunto de árboles que puede estar vacío, o puede contener n árboles (con $n > 0$).

Pregunta **3**

Correcta

Se puntúa 1 sobre 1

¿Cuáles de los siguientes son **factores esenciales** a tener en cuenta cuando se realiza el *análisis de la eficiencia de un algoritmo* (por ejemplo, para efectuar una comparación de rendimiento entre dos algoritmos que resuelven el mismo problema)?

Seleccione una o más de una:

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

Pregunta **4**

Correcta

Se puntúa 1 sobre 1

Suponga que la versión recursiva de la función para el cálculo del factorial que se presentó en clases fuese redefinida en la forma siguiente:

```
def factorial(n):  
    f = n * factorial(n - 1)  
    if n == 0:  
        return 1  
    return f
```

¿Cuál es el problema (si lo hubiese) con esta versión de la función?

Seleccione una:

- ☐ a. No hay ningún problema. Funciona correctamente.
- ☐ b. La función siempre retorna un 1, sin importar cual sea el valor de n.
- ☒ c. Contiene todos los elementos que debería tener una función recursiva bien planteada, pero en el orden incorrecto: la condición para chequear si n es cero debería estar antes que la llamada recursiva para evitar la recursión infinita. ✓
- ☐ d. Para calcular el factorial necesita un ciclo for que genere los números a multiplicar. Así planteada, la función hace un único primer cálculo incompleto y lo retorna.

Pregunta 5

Correcta

Se puntúa 2 sobre 2

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

Pregunta 6

Correcta

Se puntúa 2 sobre 2

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 $\text{producto}(a, b)$ [$a \geq 0, b \geq 0$], que podría ser la que sigue:

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

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

¿Es correcto el siguiente planteo de la función $\text{producto}(a, b)$?

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

Seleccione una:

- ☐ a. No. La propia definición previa del concepto de producto es incorrecta: un producto es una multiplicación, y no una suma...
- ☒ b. Sí. Es correcto. ✓
- ☐ c. No. La función sugerida siempre retornará 0, sean cuales fuesen los valores de a y b .
- ☐ d. No. La última línea debería decir $\text{return } a + \text{producto}(a-1, b-1)$ en lugar de $\text{return } a + \text{producto}(a, b-1)$.

Pregunta 7

Correcta

Se puntúa 2 sobre 2

En la tabla siguiente se muestra un resumen comparativo entre las versiones iterativa (basada en ciclos) y recursiva del algoritmo de cálculo del término n -ésimo de la Sucesión de [Fibonacci](#). Note que se han dejado sin completar los casilleros que corresponden al tiempo de ejecución para ambos casos.

Cálculo del término n -ésimo de Fibonacci [F(n)]	Complejidad código fuente	Consumo de memoria	Tiempo de ejecución
Versión iterativa	Aceptable	Constante (no depende de n)	???
Versión recursiva	Optima	Proporcional a n (lineal)	???

¿Cuál es la estimación para el tiempo de ejecución de ambos algoritmos?

Seleccione una:

- ☒ a. Versión iterativa: tiempo proporcional a n (lineal) - Versión recursiva: tiempo proporcional a b^n (exponencial, para algún $b > 1$) ✓
- ☐ b. Versión iterativa: tiempo proporcional a n (lineal) - Versión recursiva: tiempo proporcional a n (lineal)
- ☐ c. Versión iterativa: tiempo constante (no depende de n) - Versión recursiva: tiempo proporcional a b^n (exponencial, para algún $b > 1$)
- ☐ d. Versión iterativa: tiempo proporcional a b^n (exponencial, para algún $b > 1$) - Versión recursiva: tiempo proporcional a n (lineal)



Pregunta **8**

Correcta

Se puntúa 1 sobre 1

¿Cuáles de las siguientes propuestas generales son **ciertas** en relación al segmento de memoria conocido como **Stack Segment**?

Seleccione una o más de una:

- ☒ a. El Stack Segment funciona como una pila (o apilamiento) de datos (modalidad **LIFO**: Last In - First Out): el último dato en llegar, , se almacena en la cima del stack, y será por eso el primero en ser retirado.
- ☒ b. El Stack Segment se utiliza como soporte interno en el proceso de invocación a funciones (**con o sin recursividad**): se va llenando  a medida que se desarrolla la cascada de invocaciones, y se vacía a medida que se produce el proceso de regreso o vuelta atrás.
- ☐ c. El Stack Segment funciona como una cola (o fila) de datos (modalidad **FIFO**: First In - First Out): el primer dato en llegar, se almacena al frente del stack, y será por eso el primero en ser retirado.
- ☐ d. El Stack Segment se utiliza como soporte interno **solamente** en el proceso de invocación a funciones recursivas: se va llenando a medida que la cascada recursiva se va desarrollando, y se vacía a medida que se produce el proceso de regreso o vuelta atrás.

Pregunta 9

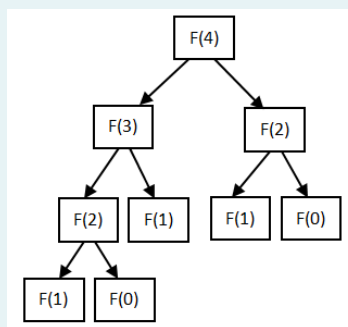
Correcta

Se puntúa 2 sobre 2

Sabemos que la recursión es una de las técnicas o estrategias básicas para el planteo de algoritmos y que una de sus ventajas es que permite el diseño de algoritmos compactos y muy claros, aunque al costo de usar algo de memoria extra en el stack segment y por consiguiente también un algo de tiempo extra por la gestión del stack. Algunos problemas pueden plantearse en forma directa procesando recursivamente diversos subproblemas menores. Tal es el caso de la *sucesión de Fibonacci*, en la cual cada término se calcula como la suma de los dos inmediatamente anteriores [esto se expresa con la siguiente relación de recurrencia: $F(n) = F(n-1) + F(n-2)$ con $F(0) = 0$ y $F(1) = 1$]. La función sencilla que mostramos a continuación que calcula el término n-ésimo de la sucesión en forma recursiva:

```
def fibo(n):
    if n <= 1:
        return 1
    return fibo(n-1) + fibo(n-2)
```

Sin embargo, un inconveniente adicional es que la *aplicación directa* de *dos o más invocaciones recursivas* en el planteo de un algoritmo podría hacer que un mismo subproblema se resuelva más de una vez, incrementando el tiempo total de ejecución. Por ejemplo, para calcular $\text{fibo}(4)$ el árbol de llamadas recursivas es el siguiente:



y puede verse que en este caso, se calcula **2 veces** el valor de $\text{fibo}(2)$, **3 veces** el valor de $\text{fibo}(1)$ y **2 veces** el valor de $\text{fibo}(0)$... con lo que la cantidad de llamadas a funciones para hacer *más de una vez el mismo trabajo* es de **7** ($= 2 + 3 + 2$).

Suponga que se quiere calcular el valor del sexto término de la sucesión. Analice el árbol de llamadas recursivas que se genera al hacer la invocación $t = \text{fibo}(6)$ **¿Cuántas veces en total la función $\text{fibo}()$ se invoca para hacer más de una vez el mismo trabajo, SIN incluir en ese conteo a las invocaciones para obtener $\text{fibo}(0)$ y $\text{fibo}(1)$?**

Seleccione una:

- ☐ a. 0
- ☐ b. 11
- ☒ c. 10 ✓
- ☐ d. 12

Pregunta 10

Correcta

Se puntúa 1 sobre 1

¿En cuáles de las siguientes situaciones el uso de *recursividad* está efectivamente recomendado? (Más de una respuesta puede ser válida. Marque todas las que considere correctas).

Seleccione una o más de una:

- ☒ a. Siempre que se pueda escribir una definición recursiva del problema. ✗
- ☐ b. Nunca.
- ☒ c. Recorrido y procesamiento de estructuras de datos no lineales como árboles y grafos. ✓
- ☒ d. Generación y procesamiento de imágenes y gráficos fractales (figuras compuestas por versiones más simples de la misma figura original). ✓

Pregunta **11**

Correcta

Se puntúa 1 sobre 1

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

Seleccione una o más de una:

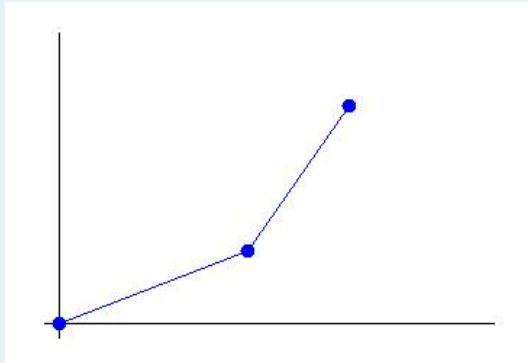
- ☒ a. El origen del sistema de coordenadas se encuentra en el punto superior izquierdo de la pantalla o de la ventana que se esté usando. ✓
- ☐ b. El origen del sistema de coordenadas se encuentra en el punto inferior izquierdo de la pantalla o de la ventana que se esté usando.
- ☒ c. Las filas de pantalla 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. ✓
- ☐ d. 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.

Pregunta 12

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar en Python el esquema de la **gráfica de una función**, como la que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la función **function(canvas)** sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def function(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucion en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, bg='white', width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    function(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función **function(canvas)** permitiría dibujar la gráfica sugerida?

Seleccione una:

☐ a.


```
def function(canvas):
    # los ejes...
    canvas.create_line((90, 400, 400, 400), fill='orange')
    canvas.create_line((100, 410, 100, 200), fill='orange')

    # las curvas...
    canvas.create_line((100, 400, 230, 350), fill='black')
    canvas.create_line((230, 350, 300, 250), fill='black')

    # los puntos...
    canvas.create_oval((95, 395, 105, 405), outline='blue', fill='blue')
    canvas.create_oval((225, 345, 235, 355), outline='blue', fill='blue')
    canvas.create_oval((295, 245, 305, 255), outline='blue', fill='blue')
```

☐ b.

```
def function(canvas):
    # los ejes...
    canvas.create_line((90, 400, 400, 400), fill='black')
    canvas.create_line((100, 410, 100, 200), fill='black')

    # las curvas...
    canvas.create_line((100, 400, 230, 350), fill='black', width=3, dash=(5, 4))
    canvas.create_line((230, 350, 300, 250), fill='black', width=3, dash=(5, 4))

    # los puntos...
    canvas.create_oval((95, 395, 105, 405), outline='blue', fill='blue')
    canvas.create_oval((225, 345, 235, 355), outline='blue', fill='blue')
    canvas.create_oval((295, 245, 305, 255), outline='blue', fill='blue')
```

☐ c.

```
def function(canvas):
    # los ejes...
    canvas.create_line((90, 400, 400, 400), fill='black')
    canvas.create_line((100, 410, 100, 200), fill='black')

    # las curvas...
    canvas.create_line((100, 400, 230, 350), fill='black')
    canvas.create_line((230, 350, 300, 250), fill='black')

    # los puntos...
    canvas.create_oval((95, 395, 105, 405), outline='blue', fill='red')
    canvas.create_oval((225, 345, 235, 355), outline='blue', fill='red')
    canvas.create_oval((295, 245, 305, 255), outline='blue', fill='red')
```

☒ d.

```
def function(canvas):
    # los ejes...
    canvas.create_line((90, 400, 400, 400), fill='black')
    canvas.create_line((100, 410, 100, 200), fill='black')

    # las curvas...
    canvas.create_line((100, 400, 230, 350), fill='blue')
    canvas.create_line((230, 350, 300, 250), fill='blue')

    # los puntos...
    canvas.create_oval((95, 395, 105, 405), outline='blue', fill='blue')
    canvas.create_oval((225, 345, 235, 355), outline='blue', fill='blue')
    canvas.create_oval((295, 245, 305, 255), outline='blue', fill='blue')
```



Pregunta **13**

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar Python *el dibujo de un botón para una aplicación que simule una interfaz visual de usuario*, como el que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la función **button(canvas)** sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def button(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucion en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    button(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función **button(canvas)** permitiría dibujar la gráfica sugerida?

Seleccione una:

- ☒ a.

```
def button(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del botón...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='light gray')
    canvas.create_text(x + ancho//2 - 1, y + alto//2 + 1, text='Ok', fill='black')

    for f in range(2):
        # los bordes blancos...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='white')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='white')

        # los bordes oscuros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='dark gray')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='dark gray')
```



☐ b.

```
def button(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del botón...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='light gray')
    canvas.create_text(x + ancho//2 - 1, y + alto//2 + 1, text='Exit', fill='black')

    for f in range(2):
        # los bordes blancos...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='white')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='white')

        # los bordes oscuros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='dark gray')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='dark gray')
```

☐ c.

```
def button(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del botón...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='light gray')
    canvas.create_text(x + ancho//2 - 1, y + alto//2 + 1, text='Ok', fill='black')

    for f in range(2):
        # los bordes blancos...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='dark gray')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='dark gray')

        # los bordes oscuros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='dark gray')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='dark gray')
```

☐ d.

```
def button(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del botón...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='red')
    canvas.create_text(x + ancho//2 - 1, y + alto//2 + 1, text='Ok', fill='black')

    for f in range(2):
        # los bordes blancos...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='white')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='white')

        # los bordes oscuros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='dark gray')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='dark gray')
```

Pregunta **14**

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar Python el dibujo *de un campo de edición de textos* para una aplicación que simule una interfaz visual de usuario, como el que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la función **edit(canvas)** sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def edit(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucion en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    edit(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función **edit(canvas)** permitiría dibujar la gráfica sugerida?

Seleccione una:

- ☒ a.

```
def edit(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del campo de edición...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='white')
    canvas.create_text(x-25, y+alto//2, text='Valor: ', fill='black')

    for f in range(2):
        # los bordes oscuros...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='dark gray')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='dark gray')

        # los bordes claros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='white')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='white')
```

- ☐ b.

```
def edit(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del campo de edición...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='white')
    canvas.create_text(x-25, y+alto//2, text='Valor: ', fill='black')

    for f in range(2):
        # los bordes oscuros...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='white')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='white')

        # los bordes claros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='dark gray')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='dark gray')
```

☐ c.

```
def edit(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del campo de edición...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='white')
    canvas.create_text(x-25, y+alto//2, text='Valor: ', fill='black')

    for f in range(6):
        # los bordes oscuros...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='dark gray')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='dark gray')

        # los bordes claros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='white')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='white')
```

☐ d.

```
def edit(canvas):
    x, y, ancho, alto = 100, 100, 100, 25

    # el fondo y el texto del campo de edición...
    canvas.create_rectangle((x, y, x+ancho, y+alto), fill='white')
    canvas.create_text(x-25, y+alto//2, text='Valor: ', fill='red')

    for f in range(2):
        # los bordes oscuros...
        canvas.create_line((x+f, y+f, x+ancho-f, y+f), fill='dark gray')
        canvas.create_line((x+f, y+f, x+f, y+alto-f), fill='dark gray')

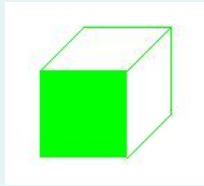
        # los bordes claros...
        canvas.create_line((x+f, y+alto-f, x+ancho-f, y+alto-f), fill='white')
        canvas.create_line((x+ancho-f, y+alto-f, x+ancho-f, y+f), fill='white')
```

Pregunta **15**

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar Python *el dibujo de un cubo* como el que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la función **cube(canvas)** sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def cube(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucion en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, bg='white', width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    cube(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función **cube(canvas)** permitiría dibujar la gráfica sugerida?

Seleccione una:

☐ a.

```
def cube(canvas):
    x, y, ancho, alto = 100, 100, 60, 60
    canvas.create_rectangle((x, y, x+ancho, y+alto), outline='navy', fill='navy')
    canvas.create_line((100, 100, 130, 70), fill='lime')
    canvas.create_line((160, 100, 190, 70), fill='lime')
    canvas.create_line((130, 70, 190, 70), fill='lime')
    canvas.create_line((190, 70, 190, 130), fill='lime')
    canvas.create_line((160, 160, 190, 130), fill='lime')
```

☐ b.

```
def cube(canvas):
    x, y, ancho, alto = 100, 100, 60, 60
    canvas.create_rectangle((x, y, x+ancho, y+alto), outline='lime', fill='lime')
    canvas.create_line((100, 100, 130, 70), fill='lime')
    canvas.create_line((160, 100, 190, 70), fill='lime')
    canvas.create_line((160, 160, 190, 130), fill='lime')
```

☒ c. ✔

```
def cube(canvas):
    x, y, ancho, alto = 100, 100, 60, 60
    canvas.create_rectangle((x, y, x+ancho, y+alto), outline='lime', fill='lime')
    canvas.create_line((100, 100, 130, 70), fill='lime')
    canvas.create_line((160, 100, 190, 70), fill='lime')
    canvas.create_line((130, 70, 190, 70), fill='lime')
    canvas.create_line((190, 70, 190, 130), fill='lime')
    canvas.create_line((160, 160, 190, 130), fill='lime')
```

☐ d.

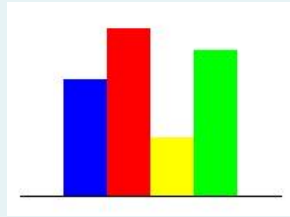
```
def cube(canvas):
    x, y, ancho, alto = 100, 100, 60, 60
    canvas.create_rectangle((x, y, x+ancho, y+alto), outline='lime', fill='lime', stipple='gray12')
    canvas.create_line((100, 100, 130, 70), fill='lime')
    canvas.create_line((160, 100, 190, 70), fill='lime')
    canvas.create_line((130, 70, 190, 70), fill='lime')
    canvas.create_line((190, 70, 190, 130), fill='lime')
    canvas.create_line((160, 160, 190, 130), fill='lime')
```

Pregunta **16**

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar en Python un esquema estadístico tipo *diagrama de barras*, como el que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la *bars(canvas)* sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def bars(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucion en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, bg='white', width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    bars(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función *bars(canvas)* permitiría dibujar la gráfica sugerida?

Seleccione una:

☐ a.

```
def bars(canvas):
    canvas.create_rectangle((100, 120, 130, 200), outline='blue')
    canvas.create_rectangle((130, 85, 160, 200), outline='red')
    canvas.create_rectangle((161, 160, 190, 200), outline='yellow')
    canvas.create_rectangle((190, 100, 220, 200), outline='lime')
    canvas.create_line((70, 200, 250, 200), fill='black')
```

☐ b.

```
def bars(canvas):
    canvas.create_rectangle((100, 120, 130, 200), outline='red', fill='red')
    canvas.create_rectangle((130, 85, 160, 200), outline='yellow', fill='yellow')
    canvas.create_rectangle((161, 160, 190, 200), outline='lime', fill='lime')
    canvas.create_rectangle((190, 100, 220, 200), outline='blue', fill='blue')
    canvas.create_line((70, 200, 250, 200), fill='black')
```


☐ c. `def bars(canvas):`
 `canvas.create_rectangle((100, 120, 130, 200), outline='blue', fill='blue')`
 `canvas.create_rectangle((130, 120, 160, 200), outline='red', fill='red')`
 `canvas.create_rectangle((161, 120, 190, 200), outline='yellow', fill='yellow')`
 `canvas.create_rectangle((190, 120, 220, 200), outline='lime', fill='lime')`
 `canvas.create_line((70, 200, 250, 200), fill='black')`

☒ d. `def bars(canvas):`
 `canvas.create_rectangle((100, 120, 130, 200), outline='blue', fill='blue')`
 `canvas.create_rectangle((130, 85, 160, 200), outline='red', fill='red')`
 `canvas.create_rectangle((161, 160, 190, 200), outline='yellow', fill='yellow')`
 `canvas.create_rectangle((190, 100, 220, 200), outline='lime', fill='lime')`
 `canvas.create_line((70, 200, 250, 200), fill='black')`

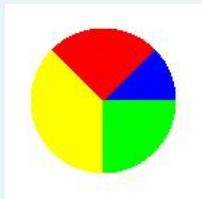


Pregunta **17**

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar en Python un esquema estadístico tipo *diagrama de segmentos circulares*, como el que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la *pie(canvas)* sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def pie(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucioen en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, bg='white', width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    pie(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función *pie(canvas)* permitiría dibujar la gráfica sugerida?

Seleccione una:

☒ a. `def pie(canvas):`
`x, y, ancho, alto = 100, 100, 100, 100`
`canvas.create_arc((x, y, x+ancho, y+alto), start=0, extent=45, outline='blue', fill='blue', style=PIESLICE)`
`canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=90, outline='red', fill='red', style=PIESLICE)`
`canvas.create_arc((x, y, x+ancho, y+alto), start=135, extent=135, outline='yellow', fill='yellow',`
`style=PIESLICE)`
`canvas.create_arc((x, y, x+ancho, y+alto), start=270, extent=90, outline='lime', fill='lime', style=PIESLICE)`



☐ b. `def pie(canvas):`
`x, y, ancho, alto = 100, 100, 100, 100`
`canvas.create_arc((x, y, x+ancho, y+alto), start=0, extent=45, outline='blue', style=PIESLICE)`
`canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=90, outline='red', style=PIESLICE)`
`canvas.create_arc((x, y, x+ancho, y+alto), start=135, extent=135, outline='yellow', style=PIESLICE)`
`canvas.create_arc((x, y, x+ancho, y+alto), start=270, extent=90, outline='lime', style=PIESLICE)`

☐ c. `def pie(canvas):`
 `x, y, ancho, alto = 100, 100, 100, 100`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=0, extent=45, outline='blue', fill='blue', style=ARC)`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=90, outline='red', fill='red', style=ARC)`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=135, extent=135, outline='yellow', fill='yellow', style=ARC)`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=270, extent=90, outline='lime', fill='lime', style=ARC)`

☐ d. `def pie(canvas):`
 `x, y, ancho, alto = 100, 100, 300, 100`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=0, extent=45, outline='blue', fill='blue', style=PIESLICE)`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=90, outline='red', fill='red', style=PIESLICE)`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=135, extent=135, outline='yellow', fill='yellow', style=PIESLICE)`
 `canvas.create_arc((x, y, x+ancho, y+alto), start=270, extent=90, outline='lime', fill='lime', style=PIESLICE)`

Pregunta **18**

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar en Python el dibujo del *esquema básico de una cara*, como el que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la *face(canvas)* sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def face(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucioen en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, bg='white', width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    face(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función *face(canvas)* permitiría dibujar la gráfica sugerida?

Seleccione una:

- ☒ a.

```
def face(canvas):
    canvas.create_oval((100, 100, 160, 160), outline='red')
    canvas.create_oval((110, 115, 125, 125), outline='blue', fill='blue')
    canvas.create_oval((135, 115, 150, 125), outline='blue', fill='blue')
    canvas.create_line((127, 134, 133, 134), fill='blue')
    canvas.create_arc((120, 138, 140, 148), start=180, extent=180, outline='blue', style=ARC)
```

 ✓
- ☐ b.

```
def face(canvas):
    canvas.create_oval((100, 100, 160, 160), outline='red')
    canvas.create_oval((110, 115, 125, 125), outline='blue', fill='blue')
    canvas.create_line((127, 134, 133, 134), fill='blue')
    canvas.create_arc((120, 138, 140, 148), start=180, extent=180, outline='blue', style=ARC)
```
- ☐ c. _____

```
def face(canvas):  
    canvas.create_oval((100, 100, 160, 160), outline='red', fill='pink')  
    canvas.create_oval((110, 115, 125, 125), outline='blue', fill='blue')  
    canvas.create_oval((135, 115, 150, 125), outline='blue', fill='blue')  
    canvas.create_line((127, 134, 133, 134), fill='blue')  
    canvas.create_arc((120, 138, 140, 148), start=180, extent=180, outline='blue', style=ARC)
```

☐ d.

```
def face(canvas):  
    canvas.create_oval((100, 100, 160, 160), outline='red')  
    canvas.create_oval((110, 115, 125, 125), outline='blue', fill='blue')  
    canvas.create_oval((135, 115, 150, 125), outline='blue', fill='blue')  
    canvas.create_line((127, 134, 133, 134), fill='blue')  
    canvas.create_arc((120, 138, 140, 148), start=0, extent=180, outline='blue', style=ARC)
```

Pregunta 19

Correcta

Se puntúa 1 sobre 1

Suponga que se desea generar en Python el *dibujo del popular Pacman*, como el que se muestra aquí (con las mismas proporciones y colores, aunque sin importar las coordenadas de visualización).



El siguiente programa está pensado para generar esa gráfica, pero dejando la *pacman(canvas)* sin hacer:

```
__author__ = 'Cátedra de AED'

from tkinter import *

def pacman(canvas):
    pass

def render():
    # configuracion inicial de la ventana principal...
    root = Tk()
    root.title('Cuestionario')

    # calculo de resolucion en pixels de la pantalla...
    maxw = root.winfo_screenwidth()
    maxh = root.winfo_screenheight()

    # ajuste de las dimensiones y coordenadas de arranque de la ventana...
    root.geometry("%dx%d+%d+%d" % (maxw, maxh, 0, 0))

    # un lienzo de dibujo dentro de la ventana...
    canvas = Canvas(root, bg='white', width=maxw, height=maxh)
    canvas.grid(column=0, row=0)

    # desarrollar la gráfica...
    pacman(canvas)

    # lanzar el ciclo principal de control de eventos de la ventana...
    root.mainloop()

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

¿Cuál de las siguientes versiones de la función *pacman(canvas)* permitiría dibujar la gráfica sugerida?

Seleccione una:

☐ a.

```
def pacman(canvas):
    x, y, ancho, alto = 100, 100, 50, 50
    canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=315, outline='red', fill='red', style=PIESLICE)
    canvas.create_oval((x+10, y+10, x+2+ancho//3, y+2+alto//3), outline='white', fill='white')
    canvas.create_oval((x+16, y+16, x+ancho/4, y+alto/4), outline='black', fill='black')
```

☒ b.

```
def pacman(canvas):
    x, y, ancho, alto = 100, 100, 50, 50
    canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=315, outline='lime', fill='lime', style=PIESLICE)
    canvas.create_oval((x+10, y+10, x+2+ancho//3, y+2+alto//3), outline='white', fill='white')
    canvas.create_oval((x+16, y+16, x+ancho/4, y+alto/4), outline='black', fill='black')
```

 ✓

☐ c.

```
def pacman(canvas):  
    x, y, ancho, alto = 100, 100, 50, 50  
    canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=359, outline='lime', fill='lime', style=PIESLICE)  
    canvas.create_oval((x+10, y+10, x+2+ancho//3, y+2+alto//3), outline='white', fill='white')  
    canvas.create_oval((x+16, y+16, x+ancho/4, y+alto/4), outline='black', fill='black')
```

☐ d.

```
def pacman(canvas):  
    x, y, ancho, alto = 100, 100, 50, 50  
    canvas.create_arc((x, y, x+ancho, y+alto), start=45, extent=315, outline='lime', fill='lime', style=PIESLICE)  
    canvas.create_oval((x+10, y+10, x+2+ancho//3, y+2+alto//3), outline='red', fill='red')  
    canvas.create_oval((x+16, y+16, x+ancho/4, y+alto/4), outline='black', fill='black')
```

[◀ Materiales Adicionales para la Ficha 14](#)

Ir a...



[Guía 14 de Ejercicios Prácticos ▶](#)