Comenzado el	viernes, 10 de mayo de 2024, 10:52
Estado	Finalizado
Finalizado en	viernes, 10 de mayo de 2024, 11:37
Tiempo empleado	44 minutos 55 segundos
Calificación	Sin calificar aún

Pregunta **1**

Finalizado

Puntúa como 1

Generar una lista por compresión que contenga la raíz cuadrada de todos los números desde 0 hasta un número limite N.

lista = [x**0.5 for x in range(n+1]

Pregunta **2**

Correcta

Se puntúa 1 sobre 1

Dadas las siguientes funciones, cuál es el resultado de ejecutar las funciones **ExVar1()**, **ExVar2()**, y **ExVar3()**.

*Nota: el orden de ejecución de las funciones es **ExVar1()**, **ExVar2()**, y **ExVar3()**.*

```
x,y = 1,2
def ExVar1():
    x = 2
    print(x,y)

def ExVar2():
    x, y = 0, 0
    def ExVar21():
        global x
        nonlocal y
        x, y = y, x
    def ExVar22():
        nonlocal x
        ExVar21()
    ExVar22()
    print(x,y)

def ExVar3():
    x = 0
    def ExVar31():
        nonlocal x
        global y
        y, x = x, y
    ExVar31()
    print(x,y)
```

ExVar1() 2 2 ✔

ExVar2() 0 1 ✔

ExVar3() 2 0 ✔

Respuesta correcta

La respuesta correcta es:

ExVar1()
→ 2 2,

ExVar2()
→ 0 1,

ExVar3()
→ 2 0

Pregunta **3**

Finalizado

Puntúa como 1

Implementar una **función iterativa** que calcule la suma de todos los números enteros comprendidos entre cero y un número entero positivo dado.

```
def suma_enteros👎 :
    suma = 0
    for i in range(n+1):
        suma +=1
    return suma
```

Pregunta **4**

Finalizado

Puntúa como 1

Implementar una **función recursiva** que calcule la suma de todos los números enteros comprendidos entre cero y un número entero positivo dado.

```
def suma_enteros_recursivo👎 :
    if n == 0:
        resultado = 0
    else:
        resultado = n + suma_enteros_recursivo(n-1)
    return resultado
```

Pregunta **5**

Finalizado

Puntúa como 1

Escribir una función que reciba dos parámetros: (i) una lista desordenada y (ii) una expresión (una expresión es parámetro que puede ser evaluado a **True** o **False**). Si el valor de la expresión es Verdadera (**True**), la lista se ordenara en forma descendente, en otro caso la función retorna una copia de la lista original. Por defecto, si la función es llamada sin una "expresión" (solo la lista) debe retornar una lista copia de la lista.

```
def ordenar_lista(lista, expresion=None):
    if expresion is None:
        resultado = lista.copy()
    else:
        if expresion:
            resultado = sorted(lista, reverse=True)
        else:
            resultado = lista.copy()
    return resultado
```

Pregunta **6**

Finalizado

Puntúa como 4

Definir una clase **Poligono**. Un **polígono** es una figura geométrica plana compuesta por una secuencia finita de lados consecutivos que encierran una región en el plano. Referencia: https://es.wikipedia.org/wiki/Pol%C3%ADgono

La clase debe contener métodos para facilitar:
- En el **__init__** del objeto deben comprobar que el polígono "cierra", es decir, la figura geométrica es cerrada.
- El ploígono es regular.
- El área del polígono.
- El perimétro del polígono.

Importante:
- Pueden agregar más atributos y métodos, si lo consideran necesario.

```
import math
class Poligono():
    def __init__(self, lados):
        self._lados = lados
        self._num_lados = len(lados)
        self._es_cerrado = self._verificar_cierre()
        self._es_regular = self._verificar_regularidad()

    def _verificar_cierre(self):
        return sum(self._lados[:-1]) > self._lados[-1]

    def _verificar_regularidad(self):
        primer_lado = self._lados[0]
        for lado in self._lados:
            if lado != primer_lado:
                return False
```

```
        return True

    def calcular_perimetro(self):
        return sum(self._lados)

    def calcular_area(self):
        if self._num_lados < 3 or not self._es_cerrado:
            return "El polígono no es cerrado o tiene menos de 3 lados. No se puede calcular el área."
        elif not self._es_regular:
            return "No se puede calcular el área porque el polígono es irregular"
        else:
            lado = self._lados[0]
            distancia_centro_lado = lado / (2 * math.tan(math.pi / self._num_lados))
            return 0.5 * self._num_lados * lado * distancia_centro_lado
```

Pregunta **7**

Finalizado

Puntúa como 1

Sobrecargar el método__**eq**__ de la clase **Poligono**. Es decir, dadas dos figuras del tipo polígono, comparar si son iguales.

```
class Poligono():
    def __init__(self, lados):
        self._lados = lados
        self._num_lados = len(lados)


    def __eq__(self, otro_poligono):
        if type(self) is type(otro_poligono):
            return self._lados == otro_ploigono._lados
        return False
```

Seguinos!

Contacto

Mitre 1399, Adrogué

campus@unab.edu.ar

Descargar la app para dispositivos móviles