Estructuras repetitivas

Alumno: Puma Huanca Anthony Rusbel
Docente: Ing. Coyla Idme Leonel
Lenguajes de programación II – FINESI
Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática

Estructuras repetitivas

Descripción

En Python, las estructuras repetitivas, también llamadas bucles o *loops*, son instrucciones que permiten ejecutar un bloque de código varias veces, dependiendo de una condición o una secuencia de elementos. Los tipos de estructuras repetitivas en Python son:

`while`: ejecuta un bloque de código mientras una condición sea verdadera.

`for`: se utiliza para iterar una secuencia como una lista, tupla, cadena o rango de números.

Ejercicio 1

Determinar el factorial de un número.
Clase: `Factorial`.
Atributo: Número.
Acción: Calcular.
Objeto: `miFactorial = Factorial(5)`

```
class Factorial:
    def __init__(self, numero):
        self.numero = numero
        self.resultado = 1

    def calcular(self):
        if self.numero < 0:
            print("El factorial no esta definido para n mero
               negativos")
            return None

        for i in range(1, self.numero + 1):
            self.resultado *= i
```

```
        return self.resultado

def main():
    miFactorial = Factorial(5)
    resultado = miFactorial.calcular()
    if resultado is not None:
        print(f"El factorial de {miFactorial.numero} es: {resultado}
            ")

if __name__ == "__main__":
    main()
```

Listing 1: Código Ejercicio 1

Ejecución:

```
El factorial de 5 es: 120
```

Ejercicio 2

Determinar la serie de Fibonacci.
Clase: `Fibonacci`.
Atributos: cantidad, serie.
Acción: generarSerie.
Objeto: `miFibonacci = Fibonacci(10)`

```
class Fibonacci:
    def __init__(self, cantidad):
        self.cantidad = cantidad
        self.serie = []

    def generarSerie(self):
        a, b = 0, 1
        for _ in range(self.cantidad):
            self.serie.append(a)
            a, b = b, a + b
        return self.serie

def main():
    cantidad = int(input("Ingrese la cantidad de la serie: "))
    miFibonacci = Fibonacci(cantidad)
    resultado = miFibonacci.generarSerie()
    print(resultado)

if __name__ == "__main__":
    main()
```

Listing 2: Código Ejercicio 2

Ejecución:

```
Ingrese la cantidad de la serie: 10
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Ejercicio 3

Crear una tabla de multiplicar.
Clase: `Tabla`.
Atributo: Número.
Acción: Multiplicación.
Objeto: `Resultado`

```python
class Tabla:
    def __init__(self, numero):
        self.numero = numero

    def multiplicacion(self):
        for i in range(1, 11):
            resultado = self.numero * i
            print(f"{self.numero} X {i} = {resultado}")

def main():
    numero = int(input("Ingrese un n mero: "))
    resultado = Tabla(numero)
    resultado.multiplicacion()

if __name__ == "__main__":
    main()
```

Listing 3: Código Ejercicio 3

Ejecución:

```
Ingrese un número: 8
8 X 1 = 8
8 X 2 = 16
8 X 3 = 24
8 X 4 = 32
8 X 5 = 40
8 X 6 = 48
8 X 7 = 56
8 X 8 = 64
8 X 9 = 72
8 X 10 = 80
```

Ejercicio 4

Determinar la suma de números naturales.
Clase: `SumaNaturales`.
Atributos: limite, suma.
Acción: calcularSuma().
Objeto: `miSuma = SumaNaturales(10)`

```
class SumaNaturales:
    def __init__(self, limite):
        self.limite = limite
        self.suma = 0

    def calcularSuma(self):
        for i in range(1, self.limite + 1):
            self.suma = self.suma + i
        return self.suma

def main():
    miSuma = SumaNaturales(10)
    resultado = miSuma.calcularSuma()
    print(f"La suma de los primeros {miSuma.limite} n meros
       naturales es {resultado}")

if __name__ == "__main__":
    main()
```

Listing 4: Código Ejercicio 4

Ejecución:

```
La suma de los primeros 10 números naturales es 55
```

Ejercicio 5

Determinar en una lista de números consecutivos qué números son pares o impares.
Clase: `Numeros`.
Atributo: Número.
Acción: Clasificar().
Objeto: `misNumeros = Numeros(10)`

```
class Numeros:
    def __init__(self, numero):
        self.numero = numero

    def clasificar(self):
        for i in range(0, self.numero + 1):
            if i == 0:
                print("El n mero es 0")
```

```
            elif i % 2 == 0:
                print(f"{i} es par")
            else:
                print(f"{i} es impar")

def main():
    misNumeros = Numeros(10)
    misNumeros.clasificar()

if __name__ == "__main__":
    main()
```

Listing 5: Código Ejercicio 5

Ejecución:

```
El número es 0
1 es impar
2 es par
3 es impar
4 es par
5 es impar
6 es par
7 es impar
8 es par
9 es impar
10 es par
```

Ejercicio 6

Crear un programa que permita ingresar una cierta cantidad de datos y determine el promedio, la varianza y la desviación estándar de todos los datos ingresados.

Clase: `Estadistica`.
Atributo: Cantidad.
Acción: `promedio()`, `varianza1P()`, `varianza2P()`, `varianza1M()`, `varianza2M()`.
Objeto: `misCalculos = Estadistica(cant)`

```
class Estadistica:
    def __init__(self, cantidad):
        self.cantidad = cantidad
        self.datos = []

    def promedio(self):
        suma = 0
        for i in range(1, self.cantidad + 1):
            dato = float(input(f"Dato {i}: "))
            self.datos.append(dato)
            suma += dato
```

```
12          return suma / self.cantidad
13
14      def varianza1P(self):
15          prom = sum(self.datos) / self.cantidad
16          suma_dif = 0
17          for x in self.datos:
18              suma_dif += (x - prom) ** 2
19          return suma_dif / self.cantidad
20
21      def varianza2P(self):
22          suma = sum(self.datos)
23          suma_cuadrados = 0
24          for z in self.datos:
25              suma_cuadrados += z ** 2
26          return (suma_cuadrados - (suma ** 2) / self.cantidad) / self
               .cantidad
27
28      def varianza1M(self):
29          prom = sum(self.datos) / self.cantidad
30          suma_dif = 0
31          for x in self.datos:
32              suma_dif += (x - prom) ** 2
33          return suma_dif / (self.cantidad - 1)
34
35      def varianza2M(self):
36          suma = sum(self.datos)
37          suma_cuadrados = 0
38          for z in self.datos:
39              suma_cuadrados += z ** 2
40          return (suma_cuadrados - (suma ** 2) / self.cantidad) / (
               self.cantidad - 1)
41
42  def main():
43      cant = int(input("Ingrese la cantidad de datos: "))
44      misCalculos = Estadistica(cant)
45      promedio = misCalculos.promedio()
46
47      varianza1 = misCalculos.varianza1P()
48      varianza2 = misCalculos.varianza2P()
49      varianza1M = misCalculos.varianza1M()
50      varianza2M = misCalculos.varianza2M()
51
52      print(f"El Promedio es: {promedio}")
53      print(f"La Varianza Poblacional calculada con la formula 1 es: {
           varianza1}")
54      print("Su Desviaci n Est ndar es:", varianza1 ** (1/2))
55      print(f"La Varianza Poblacional calculada con la formula 2 es: {
           varianza2}")
```

```
    print("Su Desviaci n Est ndar es:", varianza2 ** (1/2))
    print(f"La Varianza Muestral calculada con la formula 1 es: {
        varianza1M}")
    print("Su Desviaci n Est ndar es:", varianza1M ** (1/2))
    print(f"La Varianza Muestral calculada con la formula 2 es: {
        varianza2M}")
    print("Su Desviaci n Est ndar es:", varianza2M ** (1/2))

if __name__ == "__main__":
    main()
```

Listing 6: Código Ejercicio 6

Ejecución:

```
Ingrese la cantidad de datos: 10
Dato 1: 16
Dato 2: 14
Dato 3: 18
Dato 4: 15
Dato 5: 16
Dato 6: 14
Dato 7: 16
Dato 8: 18
Dato 9: 19
Dato 10: 15
El Promedio es: 16.1
La Varianza Poblacional calculada con la formula 1 es: 2.6900000000000004
Su Desviación Estándar es: 1.6401219466856727
La Varianza Poblacional calculada con la formula 2 es: 2.6900000000000093
Su Desviación Estándar es: 1.6401219466856753
La Varianza Muestral calculada con la formula 1 es: 2.988888888888889
Su Desviación Estándar es: 1.7288403019033338
La Varianza Muestral calculada con la formula 2 es: 2.988888888888899
Su Desviación Estándar es: 1.7288403019033365
```

Ejercicio 7

Imprimir números del 1 al 10, usando la estructura `while`.
Clase: `NumeroNatural`.
Atributo: Valor.
Acción: Mostrar.
Objeto: `numero = NumeroNatural`

```
class NumeroNatural:
    def __init__(self, valor):
        self.valor = valor
```

```
4
5       def mostrar(self):
6           print(f"N mero natural: {self.valor}")
7
8   def main():
9       i = 1
10      while i <= 10:
11          numero = NumeroNatural(i)
12          numero.mostrar()
13          i += 1
14
15  if __name__ == "__main__":
16      main()
```

Listing 7: Código Ejercicio 7

Ejecución:

```
Número natural: 1
Número natural: 2
Número natural: 3
Número natural: 4
Número natural: 5
Número natural: 6
Número natural: 7
Número natural: 8
Número natural: 9
Número natural: 10
```

Ejercicio 8

En una lista del 1 al 10 calcular si es múltiplo de 3, 5 o ninguno.
Clase: `NumeroMultiplo`.
Atributo: Valor.
Acción: mostrarMultiplo.
Objeto: `numero = NumeroMultiplo()`

```
1   class NumeroMultiplo:
2       def __init__(self, valor):
3           self.valor = valor
4
5       def mostrarMultiplo(self):
6           if self.valor == 0:
7               print(f"El {self.valor} es n mero nulo")
8           elif self.valor % 3 == 0 and self.valor % 5 == 0:
9               print(f"El {self.valor} es m ltiplo de 3 y de 5")
10          elif self.valor % 3 == 0:
11              print(f"El {self.valor} es m ltiplo de 3")
```

```
12          elif self.valor % 5 == 0:
13              print(f"El {self.valor} es m ltiplo de 5")
14          else:
15              print(f"El {self.valor} no es m ltiplo de 3 ni de 5")
16
17          print("*" * 60)
18
19  def main():
20      i = 0
21      while i <= 10:
22          numero = NumeroMultiplo(i)
23          numero.mostrarMultiplo()
24          i += 1
25
26  if __name__ == "__main__":
27      main()
```

Listing 8: Código Ejercicio 8

Ejecución:

```
El 0 es número nulo
************************************************************
El 1 no es múltiplo de 3 ni de 5
************************************************************
El 2 no es múltiplo de 3 ni de 5
************************************************************
El 3 es múltiplo de 3
************************************************************
El 4 no es múltiplo de 3 ni de 5
************************************************************
El 5 es múltiplo de 5
************************************************************
El 6 es múltiplo de 3
************************************************************
El 7 no es múltiplo de 3 ni de 5
************************************************************
El 8 no es múltiplo de 3 ni de 5
************************************************************
El 9 es múltiplo de 3
************************************************************
El 10 es múltiplo de 5
************************************************************
```

Ejercicio 9

Desarrolle la serie de Fibonacci utilizando la sentencia `while`.

Clase: `Fibonacci`.
Atributo: cantidad.
Acción: generarSerie.
Objeto: `miFibonacci = Fibonacci()`

```
class Fibonacci:
    def __init__(self, cantidad):
        self.cantidad = cantidad
        self.a = 0
        self.b = 1
        self.contador = 0

    def generarSerie(self):
        print("Serie de Fibonacci")
        while self.contador < self.cantidad:
            print(self.a, end=" ")
            c = self.a + self.b
            self.a = self.b
            self.b = c
            self.contador += 1

def main():
    cant = int(input("Ingrese una cantidad: "))
    miFibonacci = Fibonacci(cant)
    miFibonacci.generarSerie()

if __name__ == "__main__":
    main()
```

Listing 9: Código Ejercicio 9

Ejecución:

```
Ingrese una cantidad: 10
Serie de Fibonacci
0 1 1 2 3 5 8 13 21 34
```

Ejercicio 10

Generar números del 1 al 10 usando el ciclo `while`.
Clase: `Numeros`.
Atributo: cantidad.
Acción: imprimir.
Objeto: `miObjeto = Numeros()`

```
class Numeros:
    def __init__(self, cantidad):
        self.cantidad = cantidad
```

```
        self.contador = 1

    def imprimir(self):
        while self.contador <= 10:
            print(self.contador)
            self.contador += 1

def main():
    cantidad = int(input("Ingrese un n mero: "))
    miObjeto = Numeros(cantidad)
    miObjeto.imprimir()

if __name__ == "__main__":
    main()
```

Listing 10: Código Ejercicio 10

Ejecución:

```
Ingrese un número: 10
1
2
3
4
5
6
7
8
9
10
```

Ejercicio 11

Hacer una calculadora que sume números y pare cuando escribas `fin`.
Clase: `CalculadoraSuma`.
Atributo: total.
Acción: sumarNumeros.
Objeto: `calculadora = CalculadoraSuma()`

```
class CalculadoraSuma:
    def __init__(self):
        self.total = 0

    def sumarNumeros(self):
        print("Calcula la suma de n meros ingresados")
        print("Escribe n meros para sumar. Escribe 'fin' para
           terminar")
        entrada = ""
```

```
        while entrada.lower() != 'fin':
            entrada = input("Ingrese un n mero: ")
            if entrada.isdigit():
                self.total += int(entrada)
            elif entrada.lower() != "fin":
                print("Entrada invalida: Escriba un n mero o 'fin'"
                    )
        print(f"La suma total es: {self.total}")

def main():
    calculadora = CalculadoraSuma()
    calculadora.sumarNumeros()

if __name__ == "__main__":
    main()
```

Listing 11: Código Ejercicio 11

Ejecución:

```
Calcula la suma de números ingresados
Escribe números para sumar. Escribe 'fin' para terminar
Ingrese un número: 8
Ingrese un número: 9
Ingrese un número: 4
Ingrese un número: 5
Ingrese un número: 10
Ingrese un número: 5
Ingrese un número: fin
La suma total es: 41
```

Ejercicio 12

Verificar si un número es nulo, par o impar; escribir `'fin'` para terminar.
Clase: `Verificar`.
Atributo: `pass`.
Acción: respuesta.
Objeto: `numeros = Verificar()`

```
class Verificar:
    def __init__(self):
        pass

    def respuesta(self):
        print("Verificar si un n mero es nulo, par o impar")
        print("Escribe n meros o 'fin' para terminar")
        entrada = ""
        while entrada.lower() != "fin":
```

```
            entrada = input("Ingrese un n mero: ")
            if entrada.isdigit():
                if int(entrada) == 0:
                    print("Es nulo")
                elif int(entrada) % 2 == 0:
                    print("Es par")
                else:
                    print("Es impar")
            elif entrada.lower() != "fin":
                print("Entrada invalida: Escriba un n mero o 'fin'"
                    )

def main():
    numeros = Verificar()
    numeros.respuesta()

if __name__ == "__main__":
    main()
```

Listing 12: Código Ejercicio 12

Ejecución:

```
Verificar si un número es nulo, par o impar
Escribe números o 'fin' para terminar
Ingrese un número: 0
Es nulo
Ingrese un número: 2
Es par
Ingrese un número: 3
Es impar
Ingrese un número: 5
Es impar
Ingrese un número: 4
Es par
Ingrese un número: fin
```

Ejercicio 13

Desarrollar un gestor de tareas que tenga un menú donde se pueda escoger si agregar tareas, mostrar tareas o salir.

Clase: `GestorTareas`.
Atributo: `tareas[]`.
Acción: `agregar_tarea`, `mostrar_tareas`.
Objeto: `mi_gestor = GestorTareas()`

```
class GestorTareas:
```

```
    def __init__(self):
        self.tareas = []

    def agregar_tarea(self, tarea):
        self.tareas.append(tarea)
        print("Tarea agregada")

    def mostrar_tareas(self):
        if not self.tareas:
            print("No hay tareas pendientes")
        else:
            print("Tareas pendientes")
            for i, tarea in enumerate(self.tareas, 1):
                print(f"{i}.- {tarea}")

def main():
    mi_gestor = GestorTareas()
    while True:
        print("\n.---- MENU ----")
        print("1.- Agregar tarea:")
        print("2.- Mostrar tarea:")
        print("3.- Salir")
        opcion = input("Seleccione una opci n: ")

        if opcion == "1":
            tarea = input("Escribe la tarea: ")
            mi_gestor.agregar_tarea(tarea)

        elif opcion == "2":
            mi_gestor.mostrar_tareas()

        elif opcion == "3":
            print("Saliendo del gestor de tareas")
            break
        else:
            print("Opci n no valida. Intenta de nuevo")

if __name__ == "__main__":
    main()
```

Listing 13: Código Ejercicio 13

Ejecución:

```
.---- MENU ----
1.- Agregar tarea:
2.- Mostrar tarea:
3.- Salir
Seleccione una opción: 1
```

```
Escribe la tarea: Clases de programación
Tarea agregada

.---- MENU ----
1.- Agregar tarea:
2.- Mostrar tarea:
3.- Salir
Seleccione una opción: 1
Escribe la tarea: Almorzar
Tarea agregada

.---- MENU ----
1.- Agregar tarea:
2.- Mostrar tarea:
3.- Salir
Seleccione una opción: 1
Escribe la tarea: Realizar tareas
Tarea agregada

.---- MENU ----
1.- Agregar tarea:
2.- Mostrar tarea:
3.- Salir
Seleccione una opción: 2
Tareas pendientes
1.- Clases de programación
2.- Almorzar
3.- Realizar tareas

.---- MENU ----
1.- Agregar tarea:
2.- Mostrar tarea:
3.- Salir
Seleccione una opción: 3
Saliendo del gestor de tareas
```

Ejercicio 14

Verificar si los números ingresados están dentro del rango de 0 a 10.
Clase: `Comprobar`.
Atributo: `pass`.
Acción: `rpta()`.
Objeto: `numeros = Comprobar()`

```
1 class Comprobar:
```

```
    def __init__(self):
        pass

    def rpta(self):
        print("Verificar n meros")
        entrada = int(input("Ingrese un n mero: "))
        while entrada > 0 and entrada <= 10:
            print(f"El n mero {entrada} est  en el rango (1-10)")
            entrada = int(input("Ingrese otro n mero (fuera de
               1-10) para salir: "))

def main():
    numeros = Comprobar()
    numeros.rpta()

if __name__ == "__main__":
    main()
```

Listing 14: Código Ejercicio 14

Ejecución:

```
Verificar números
Ingrese un número: 5
El número 5 está en el rango (1-10)
Ingrese otro número (fuera de 1-10) para salir: 1
El número 1 está en el rango (1-10)
Ingrese otro número (fuera de 1-10) para salir: 8
El número 8 está en el rango (1-10)
Ingrese otro número (fuera de 1-10) para salir: 11
```