



PE1: Módulo 3: Valores booleanos, ejecución condicional, bucles, listas y su procesamiento, operadores lógicos y de bit a bit

Profesor:

Javier Jesús Torres Yañez

Unidad 1:

Principios de Programación de Redes

Alumno:

Juan Miguel Hernández Beltrán 1223100787

GRUPO:

GIR0642

TSU Infraestructura de Redes Digitales

3.1.1.4 LABORATORIO: Preguntas y respuestas

En esta actividad, aprendí a utilizar la función `input()` y a aplicar operadores de comparación de manera efectiva mediante este programa

LABORATORIO

Tiempo Estimado
5 minutos

Nivel de Dificultad
Muy Fácil

Objetivos

- Familiarizarse con la función `input()`.
- Familiarizarse con los operadores de comparación en Python.

Escenario

Usando uno de los operadores de comparación en Python, escribe un programa simple de dos líneas que tome el parámetro `n` como entrada, que es un entero, e imprime `False` si `n` es menor que 100, y `True` si `n` es mayor o igual que 100.

No debes crear ningún bloque `if` (hablaremos de ellos muy pronto). Prueba tu código usando los datos que te proporcionamos.

Datos de Prueba

```
1 #Juan Miguel Hernández Beltrán
2
3 n = int(input("Introduce un número: "))
4 print(n >= 100)
```

Console >_

Introduce un número: 55
False
Introduce un número: 99
False
Introduce un número: 100
True
Introduce un número: 101
True
Introduce un número: -5
False
Introduce un número: +123
True
Introduce un número:

3.1.1.10 LABORATORIO: Operadores de comparación y ejecución condicional

En esta actividad aprendí a implementar la ejecución condicional utilizando la función `input()`. Al crear un programa que responde a diferentes entradas relacionadas con la planta Espatifilo.

MODULE (15%)

SECTION (73%)

ingresada es "espatifilo".

- Imprima "¡ESPATIFILIO!, ¡No [entrada]!" de lo contrario. Nota: [entrada] es la cadena que se toma como entrada.

Prueba tu código con los datos que te proporcionamos. ¡Y hazte de un ESPATIFILIO también!

Datos de Prueba

Entrada de muestra: espatifilo

Resultado esperado: No, ¡quiero un gran ESPATIFILIO!

Entrada de ejemplo: pelargonio

Resultado esperado: ¡ESPATIFILIO!, ¡No pelargonio!

Entrada de muestra: ESPATIFILIO

Resultado esperado: Si, ¡El ESPATIFILIO es la mejor planta de todos los tiempos!

```
1 #Juan Miguel Hernández Beltrán
2 entrada = input("Introduce el nombre de la planta: ")
3
4 if entrada == "ESPATIFILIO":
5     print("Si, ¡El ESPATIFILIO! es la mejor planta de todos los tiempos!")
6 elif entrada == "espatifilo":
7     print("No, ¡quiero un gran ESPATIFILIO!")
8 else:
9     print(f"¡ESPATIFILIO!, ¡No {entrada}!")
10
```

Console >_

Introduce el nombre de la planta: espatifilo
No, ¡quiero un gran ESPATIFILIO!
Introduce el nombre de la planta: pelargonio
¡ESPATIFILIO!, ¡No pelargonio!
Introduce el nombre de la planta: ESPATIFILIO
Si, ¡El ESPATIFILIO! es la mejor planta de todos los tiempos!

PrevNext

Restablecer 100

3.1.1.11 LABORATORIO: Fundamentos de la sentencia if-else

En esta actividad, aprendí a utilizar la sentencia `if-else` para ramificar la lógica de un programa en Python.

Nivel de Dificultad

Fácil/Medio

Objetivos

Familiarizar al estudiante con:

- Utilizar la sentencia *if-else* para ramificar la ruta de control.
- Construir un programa completo que resuelva problemas simples de la vida real.

Escenario

Érase una vez una tierra de leche y miel, habitada por gente feliz y próspera. La gente pagaba impuestos, por supuesto, su felicidad tenía límites. El impuesto más importante, denominado *Impuesto Personal de Ingresos (IPI*, para abreviar) tenía que pagarse una vez al año y se evaluó utilizando la siguiente regla:

- Si el ingreso del ciudadano no era superior a 85,528 pesos, el impuesto era igual al 18% del ingreso menos 556 pesos y 2 centavos (esta fue la llamada *exención fiscal*).
- Si el ingreso era superior a esta cantidad, el impuesto era igual a 14,839 pesos y 2 centavos, más el 32% del excedente sobre 85,528 pesos.

Tu tarea es escribir una **calculadora de impuestos**.

```

1 #Juan Miguel Hernández Beltrán
2 income = float(input("Introduce el ingreso anual:"))
3
4 #
5 # Escribe tu código aquí.
6 # Definir el límite de 85,528 pesos
7 limite = 85528.0
8
9 # impuesto
10 if income <= limite:
11     tax = income * 0.18 - 556.02
12 else:
13     tax = 14839.02 + (income - limite) * 0.32
14
15 # Si el impuesto es menor a cero
16 if tax < 0:
17     tax = 0.0
18
19 # Redondear el impuesto y mostrar el resultado
20 tax = round(tax, 0)
21 print("El impuesto es:", tax, "pesos")

```

Console >_

```

Introduce el ingreso anual:10000
El impuesto es: 1244.0 pesos
Introduce el ingreso anual:100000
El impuesto es: 19470.0 pesos
Introduce el ingreso anual:1000
El impuesto es: 0.0 pesos
Introduce el ingreso anual:-100
El impuesto es: 0.0 pesos

```

3.1.1.12 LABORATORIO: Fundamentos de la sentencia if-elif-else

En esta actividad, aprendí a utilizar la sentencia if-elif-else para implementar reglas lógicas al desarrollar un programa que determina si un año es bisiestro o común.

Tiempo Estimado

10-15 minutos

Nivel de Dificultad

Fácil/Medio

Objetivos

Familiarizar al estudiante con:

- Utilizar la sentencia if-elif-else.
- Encontrar la implementación adecuada de las reglas definidas verbalmente.
- Emplear el código de prueba empleando entradas y salidas de muestra.

Escenario

Como seguramente sabrás, debido a algunas razones astronómicas, el año pueden ser *bisiesto* o *común*. Los primeros tienen una duración de 366 días, mientras que los últimos tienen una duración de 365 días.

Desde la introducción del calendario Gregoriano (en 1582), se utiliza la siguiente regla para determinar el tipo de año:

- Si el número del año no es divisible entre cuatro, es un *año común*.
- De lo contrario, si el número del año no es divisible entre 100, es un *año bisiesto*.
- De lo contrario, si el número del año es divisible entre 400, es un *año común*.
- De lo contrario, es un *año bisiesto*.

```

1 #Juan Miguel Hernández Beltrán
2 year = int(input("Introduce un año:"))
3
4 #
5 # Escribe tu código aquí.
6 #
7 if year < 1582:
8     print("No dentro del periodo del calendario Gregoriano")
9 else:
10     # Comprobar si el año es bisiesto o común
11     if year % 4 != 0:
12         print("Año Común")
13     elif year % 100 != 0:
14         print("Año Bisiesto")
15     elif year % 400 != 0:
16         print("Año Común")
17     else:
18         print("Año Bisiesto")
19

```

Console >_

```

Introduce un año:2000
Año Bisiesto
Introduce un año:2015
Año Común
Introduce un año:1999
Año Común
Introduce un año:1996
Año Bisiesto
Introduce un año:1580
No dentro del periodo del calendario Gregoriano

```

3.2.1.3 LABORATORIO: Lo esencial del bucle while - Adivina el número secreto

En esta actividad, aprendí a utilizar el bucle while para crear un juego interactivo en Python.

LABORATORIO

Tiempo Estimado

15 minutos

Nivel de Dificultad

Fácil

Objetivos

Familiarizar al estudiante con:

- Utilizar el bucle `while`.
- Reflejar situaciones de la vida real en código de computadora.

Escenario

Un mago junior ha elegido un número secreto. Lo ha escondido en una variable llamada `secret_number`. Quiere que todos los que ejecutan su programa jueguen el juego *Adivina el número secreto*, y adivina qué número ha elegido para ellos. ¡Quiénes no adivinen el número quedarán atrapados en un bucle sin fin para siempre! Desafortunadamente, él no sabe cómo completar el código.

Tu tarea es ayudar al mago a completar el código en el editor de tal manera que el código:

- Pedirá al usuario que ingrese un número entero.
- Utilizará un bucle `while`.
- Comprobará si el número ingresado por el usuario es el mismo que el número escogido por el mago. Si el número elegido por el usuario es diferente al número secreto del mago, el usuario debería ver el mensaje `"¡Ja, ja! ¡Estás atrapado en mi bucle!"` y se le solicitará que ingrese un número nuevamente. Si el número ingresado por el usuario coincide con el número escogido por el mago, el número debe imprimirse en la pantalla, y el mago debe decir las siguientes palabras: `"¡Bien hecho, muggle! Eres libre ahora"`.

¡El mago está contando contigo! No lo decepciones.

INFO EXTRA

Por cierto, observa la función `print()`. La forma en que lo hemos utilizado aquí se llama *impresión multiínea*. Puede utilizar *comillas triples* para imprimir cadenas en varias líneas para facilitar la lectura del texto o crear un

```
1 #Juan Miguel Hernández Beltrán
2 secret_number = 777
3
4 print(
5     """
6     ¡Bienvenido a mi juego, muggle!
7     Introduce un número entero
8     y adivina qué número he
9     elegido para ti.
10    Entonces,
11    ¿Cuál es el número secreto?
12    """
13 )
14
15 while True:
16     user_guess = int(input("Introduce tu número: "))
17     if user_guess == secret_number:
18         print("¡Bien hecho, muggle! Eres libre ahora.")
19         break # Salir del bucle si el número es correcto
20     else:
21         print("¡Ja, ja! ¡Estás atrapado en mi bucle!")
22
23
24
```

Console >_

```
=====
¡Bienvenido a mi juego, muggle!
Introduce un número entero
y adivina qué número he
elegido para ti.
Entonces,
¿Cuál es el número secreto?
=====
Introduce tu número: 700
¡Ja, ja! ¡Estás atrapado en mi bucle!
Introduce tu número: 777
¡Bien hecho, muggle! Eres libre ahora.
```

3.2.1.6 LABORATORIO: Fundamentos del bucle for: el conteo

En esta actividad, aprendí a utilizar un bucle for para implementar un conteo secuencial al crear un programa que cuenta

LABORATORIO

Tiempo Estimado

5 minutos

Nivel de dificultad

Muy fácil

Objetivos

Familiarizar al estudiante con:

- Utilizar el bucle `for`.
- Reflejar situaciones de la vida real en código de computadora.

Escenario

¿Sabes lo que es Mississippi? Bueno, es el nombre de uno de los estados y ríos en los Estados Unidos. El río Mississippi tiene aproximadamente 2,340 millas de largo, lo que lo convierte en el segundo río más largo de los Estados Unidos (el más largo es el río Missouri). ¡Es tan largo que una sola gota de agua necesita 90 días para recorrer toda su longitud!

La palabra *Mississippi* también se usa para un propósito ligeramente diferente: para *contar mississippily* (*mississippimente*).

Si no estás familiarizado con la frase, estamos aquí para explicarte lo que significa: se utiliza para contar segundos.

```
1 #Juan Miguel Hernández Beltrán
2 import time
3
4 # Escribe un bucle for que cuente hasta cinco.
5 # Cuerpo del bucle: imprime el número de iteración del bucle y la palabra "Mississippi".
6 # Cuerpo del bucle - usar: time.sleep (1)
7
8 # Escribe una función de impresión con el mensaje final.
9
10
11 # Bucle que cuenta hasta cinco con la palabra "Mississippi"
12 for i in range(1, 6):
13     print(f"¡{i} Mississippi")
14     time.sleep(1)
15
16 # Mensaje final
17 print("¡Listo!")
18
```

Console >_

```
1 Mississippi
2 Mississippi
3 Mississippi
4 Mississippi
5 Mississippi
¡Listo
```

3.2.1.9 LABORATORIO: La sentencia break - Atascado en un bucle

En esta actividad, aprendí a utilizar la instrucción `break` en un bucle `while` para controlar el flujo de un programa.

LABORATORIO

Tiempo Estimado

10 minutos

Nivel de Dificultad

Fácil

Objetivos

Familiarizar al estudiante con:

- Utilizar la instrucción `break` en los bucles.
- Reflejar situaciones de la vida real en código de computadora.

Escenario

La instrucción `break` se implementa para salir/terminar un bucle.

Diseña un programa que use un bucle `while` y le pida continuamente al usuario que ingrese una palabra a menos que ingrese "chupacabra" como la palabra de salida secreta, en cuyo caso el mensaje "¡Has dejado el bucle con éxito!". Debe imprimirse en la pantalla y el bucle debe terminar.

No imprimas ninguna de las palabras ingresadas por el usuario. Utiliza el concepto de ejecución condicional y la sentencia `break`.

```
1 #Juan Miguel Hernández Beltrán
2 while True:
3     palabra = input("Introduce una palabra (o la palabra secreta para salir): ")
4     if palabra == "chupacabra":
5         print("¡Has dejado el bucle con éxito!")
6         break # Termina el bucle si la palabra es "chupacabra"
7
```

Console>_

Introduce una palabra (o la palabra secreta para salir): palabra
Introduce una palabra (o la palabra secreta para salir): chupacabra
¡Has dejado el bucle con éxito!

3.2.1.10 LABORATORIO: La sentencia continue - El Feo Devorador de Vocales

En esta actividad, aprendí a utilizar la instrucción `continue` dentro de un bucle `for` para omitir ciertas iteraciones al crear un "devorador de vocales" que implementé la lógica condicional para eliminar las vocales de una palabra ingresada por el usuario

3.2.1.10 LABORATORIO: La sentencia continue - El Feo Devorador de Vocales

MODULE (33%)

SECTION (59%)

Datos de Prueba

Entrada de muestra: Gregory

Salida esperada:

G
R
G
R
Y

Entrada de muestra: abstemious

Salida esperada:

B
S
T
M
S

Entrada de muestra: IOUEA

Salida esperada:

```
1 #Juan Miguel Hernández Beltrán
2 # Pedir al usuario que ingrese una palabra
3 user_word = input("Ingresa una palabra: ").upper()
4
5 # Bucle para recorrer cada letra en la palabra ingresada
6 for letter in user_word:
7     # Si la letra es una vocal, se salta la iteración con continue
8     if letter in ['A', 'E', 'I', 'O', 'U']:
9         continue
10    # Imprimir las letras no consumidas en líneas separadas
11    print(letter)
12
```

Console>_

Ingresa una palabra: Gregory
G
R
G
R
Y
Ingresa una palabra: abstemious
B
S
T
M
S
Ingresa una palabra: IOUEA

3.2.1.11 LABORATORIO: La sentencia continue - El Bonito Devorador de Vocales

En esta actividad, aprendí a rediseñar un programa para crear un "devorador de vocales" utilizando un bucle `for` y la instrucción `continue`, implementé la lógica condicional para eliminar las vocales de una palabra ingresada por el usuario.

Python Institute

3.2.1.11 LABORATORIO: La sentencia continue - El Bonito Devorador de Vocales

MODULE (35%)

SECTION (65%)

LABORATORIO

Tiempo Estimado

10 minutos

Nivel de Dificultad

Fácil

Objetivos

Familiarizar al estudiante con:

- Utilizar la instrucción `continue` en los bucles.
- Modificar y actualizar el código existente.
- Reflejar situaciones de la vida real en código de computadora.

Escenario

La tarea aquí es aún más especial que antes: ¡Debes rediseñar el devorador de vocales del laboratorio anterior (3.1.2.10) y crear un mejor devorador de vocales (bonito) mejorado! Escribe un programa que use:

- Un bucle `for`.
- El concepto de ejecución condicional (*if-elif-else*).
- La instrucción `continue`.

```
1 #Juan Miguel Hernández Beltrán
2 word_without_vowels = ""
3
4 # Solicitar al usuario que ingrese una palabra y convertirla a mayúsculas
5 user_word = input("Ingresa una palabra: ").upper()
6
7 # Bucle para recorrer cada letra de la palabra ingresada
8 for letter in user_word:
9     # Si la letra es una vocal, se salta
10     if letter in ('A', 'E', 'I', 'O', 'U'):
11         continue
12     # Si no es vocal, se añade a word_without_vowels
13     word_without_vowels += letter
14
15 # Imprimir la palabra sin vocales
16 print(word_without_vowels)
17
```

Console>_

Ingresar una palabra: Gregory
GRGRY
Ingresar una palabra: abstemious
BSTMS
Ingresar una palabra: IOUEA

3.2.1.14 LABORATORIO: Fundamentos del bucle while

En esta actividad, aprendí a utilizar un bucle while para resolver un problema práctico relacionado con la construcción de una pirámide de bloques

Python Institute

3.2.1.14 LABORATORIO: Fundamentos del bucle while

MODULE (39%)

SECTION (82%)

LABORATORIO

Tiempo Estimado

20-30 minutos

Nivel de Dificultad

Medio

Objetivos

Familiarizar al estudiante con:

- Utilizar el bucle `while`.
- Encontrar la implementación adecuada de reglas definidas verbalmente.
- Reflejar situaciones de la vida real en código de computadora.

Escenario

Escucha esta historia: Un niño y su padre, un programador de computadoras, juegan con bloques de madera. Están construyendo una pirámide.

Su pirámide es un poco rara, ya que en realidad es una pared en forma de pirámide, es plana. La pirámide se apila de acuerdo con un principio simple: cada capa inferior contiene un bloque más que la capa superior.

```
1 #Juan Miguel Hernández Beltrán
2 blocks = int(input("Ingresa el número de bloques: "))
3
4 #
5 # Escribe tu código aquí.
6
7 height = 0
8 used_blocks = 0
9
10 # Bucle para sumar capas hasta que no haya suficientes bloques
11 while used_blocks + height + 1 <= blocks:
12     height += 1
13     used_blocks += height
14
15 # Imprimir la altura de la pirámide
16 print("La altura de la pirámide:", height)
17
```

Console>_

Ingresar el número de bloques: 6
La altura de la pirámide: 3
Ingresar el número de bloques: 20
La altura de la pirámide: 5
Ingresar el número de bloques: 1000
La altura de la pirámide: 44
Ingresar el número de bloques: 2
La altura de la pirámide: 1

3.2.1.15 LABORATORIO: Hipótesis de Collatz

En esta actividad, aprendí a implementar la hipótesis de Collatz utilizando un bucle while al desarrollar un programa que toma un número natural y aplica las reglas de la hipótesis, pude reflejar un concepto matemático en código.

5

16

8

4

2

1

pasos = 17

Entrada de muestra: 16

Salida esperada:

8

4

2

1

pasos = 4

Entrada de muestra: 1023

Salida esperada:

3070

1535

4606

2303

6910

3455

10366

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

Console>_

Ingresar un número natural: 15

46

23

70

35

106

53

160

80

40

20

10

5

16

8

4

3.2.1.15 LABORATORIO: Hipótesis de Collatz

MODULE (40%)

SECTION (88%)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

Console>_

Ingresar un número natural: 16

8

4

2

1

pasos = 4

3.2.1.15 LABORATORIO: Hipótesis de Collatz

MODULE (40%)

SECTION (88%)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

Console>_

11

34

17

52

26

13

40

20

10

5

16

8

4


2

1

pasos = 62

3.4.1.6 LABORATORIO: Lo básico de las listas




En esta actividad, aprendí a trabajar con listas lo cual me permitió crear y modificar estructuras de datos de manera mas sencilla.



3.4.1.6 LABORATORIO: Lo básico de las listas

MODULE (59%)

SECTION (43%)



Tiempo Estimado
5 minutos

Nivel de Dificultad
Muy fácil







Objetivos
Familiarizar al estudiante con:

- Usar instrucciones básicas relacionadas con listas.
- Crear y modificar listas.

Escenario
Había una vez un sombrero. El sombrero no contenía conejo, sino una lista de cinco números: 1, 2, 3, 4 y 5.

Tu tarea es:

- Escribir una línea de código que solicite al usuario que reemplace el número central en la lista con un número entero ingresado por el usuario (Paso 1).
- Escribir una línea de código que elimine el último elemento de la lista (Paso 2).
- Escribir una línea de código que imprima la longitud de la lista existente (Paso 3).




```
1 #Juan Miguel Hernández Beltrán
2 hat_list = [1, 2, 3, 4, 5] # Esta es una lista existente de números ocultos en el sombrero.
3
4
5 # Paso 1: Solicitar al usuario un número para reemplazar el valor en el centro de la lista
6 hat_list[2] = int(input("Reemplaza el número central con un número: "))
7
8 # Paso 2: Eliminar el último elemento de la lista
9 hat_list.pop()
10
11 # Paso 3: Imprimir la longitud de la lista existente
12 print("La longitud de la lista es:", len(hat_list))
13
14 # Imprimir la lista actualizada
15 print(hat_list)
16
17
```

Console >_
Reemplaza el número central con un número: 7
La longitud de la lista es: 4

3.6.1.9 LABORATORIO: Operando con listas - conceptos básicos




En esta actividad, aprendí a trabajar con la indexación de listas y a utilizar los operadores in y not in Al desarrollar un programa que elimina las repeticiones de números en una lista, creé una nueva lista para almacenar solo los elementos que sean únicos.



3.6.1.9 LABORATORIO: Operando con listas - conceptos básicos

MODULE (87%)

SECTION (90%)



Nivel de Dificultad
Fácil

Objetivos
Familiarizar al estudiante con:

- Indexación de listas.
- Utilizar operadores `in` y `not in`.







Escenario
Imagina una lista: no muy larga ni muy complicada, solo una lista simple que contiene algunos números enteros. Algunos de estos números pueden estar repetidos, y esta es la clave. No queremos ninguna repetición. Queremos que sean eliminados.

Tu tarea es escribir un programa que elimine todas las repeticiones de números de la lista. El objetivo es tener una lista en la que todos los números aparezcan no más de una vez.

Nota: Asume que la lista original está ya dentro del código, no tienes que ingresarla desde el teclado. Por supuesto, puedes mejorar el código y agregar una parte que pueda llevar a cabo una conversación con el usuario y obtener todos los datos.

Sugerencia: Te recomendamos que crees una nueva lista como área de trabajo temporal, no necesitas actualizar la lista actual.

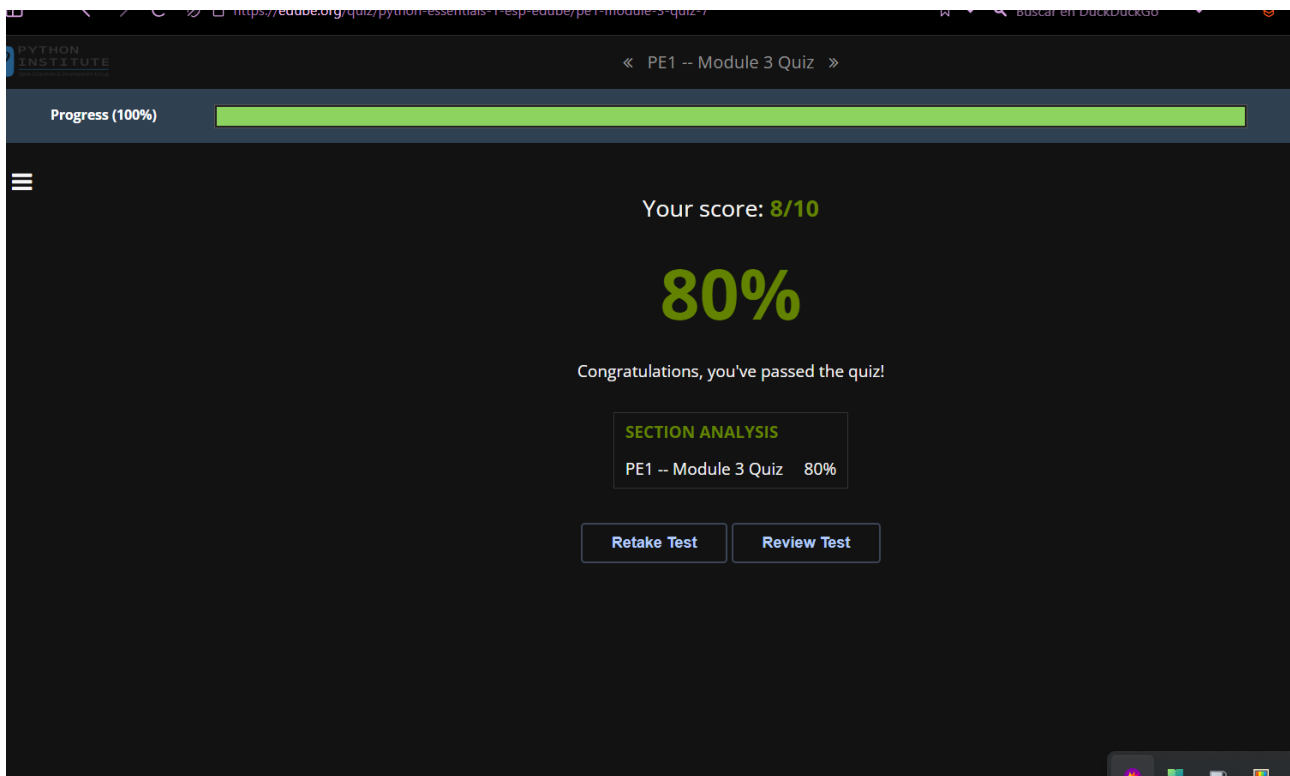
No hemos proporcionado datos de prueba, ya que sería demasiado fácil. Puedes usar



```
1 #Juan Miguel Hernández Beltrán
2 my_list = [1, 2, 4, 4, 1, 4, 2, 6, 2, 9]
3
4 # Escribe tu código aquí.
5 # Lista original con elementos repetidos
6
7 # Crear una nueva lista para almacenar elementos únicos
8 unique_list = []
9
10 # Recorrer cada elemento de la lista original
11 for number in my_list:
12     # Si el número no está en la lista única, añadirlo
13     if number not in unique_list:
14         unique_list.append(number)
15
16 # Imprimir la lista con elementos únicos
17 print("La lista con elementos únicos:")
18 print(unique_list)
19
```

Console >_
La lista con elementos únicos:
[1, 2, 4, 6, 9]

Pantalla de Quiz



Pantalla de Examen Módulo 3

