



PE1: Módulo 4: Funciones, Tuplas, Diccionarios y Procesamiento de Datos

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

4.3.1.6 LABORATORIO: Un año bisiesto: escribiendo tus propias funciones

En este laboratorio, aprendí a crear funciones, escribí una función que toma un año como argumento y devuelve True si es bisiesto, o False si no lo es. Al probarla con listas de años y resultados esperados.

Python INSTITUTE

« 4.3.1.6 LABORATORIO: Un año bisiesto: escribiendo tus propias funciones »

MODULE (30%)

SECTION (55%)

LABORATORIO

Tiempo Estimado
10-15 minutos

Nivel de Dificultad
Fácil

Objetivos
Familiarizar al estudiante con:

- Proyectar y escribir funciones con parámetros.
- Utilizar la instrucción return.
- Probar las funciones.

Escenario
Tu tarea es escribir y probar una función que toma un argumento (un año) y devuelve True si el año es un año bisiesto, o False si no lo es.

Parte del esqueleto de la función ya está en el editor.

Nota: también hemos preparado un breve código de prueba, que puedes utilizar para probar tu función.

El código utiliza dos listas: una con los datos de prueba y la otra con los resultados esperados. El código te dirá si alguno de tus resultados no es válido.

```
1 #Juan Miguel Hernández Beltrán
2 def is_year_leap(year):
3     #
4     # Escribe tu código aquí
5     if year % 4 == 0:
6         if year % 100 == 0:
7             if year % 400 == 0:
8                 return True
9             else:
10                return False
11        else:
12            return True
13    else:
14        return False
15
16 # Datos de prueba
17 test_data = [1900, 2000, 2016, 1987]
18 test_results = [False, True, True, False]
19
20 # Probar la función con los datos de prueba
21 for i in range(len(test_data)):
22     yr = test_data[i]
23     print(yr, "->", end="")
24     result = is_year_leap(yr)
25     if result == test_results[i]:
26         print("OK")
27     else:
```

Console >_

1900 ->OK
2000 ->OK
2016 ->OK
1987 ->OK

4.3.1.7 LABORATORIO: ¿Cuántos días?: escribiendo y utilizando tus propias funciones

En este laboratorio, aprendí a escribir funciones parametrizadas y Desarrollé una función que toma un año y un mes como argumentos y devuelve el número de días en ese mes, considerando si el año es bisiesto. Implementé validaciones para devolver None si los argumentos no son válidos.

Python INSTITUTE

« 4.3.1.7 LABORATORIO: ¿Cuántos días?: escribiendo y utilizando tus propias funciones »

MODULE (22%)

SECTION (64%)

LABORATORIO

Tiempo Estimado
15-20 minutos

Nivel de Dificultad
Medio

Requisitos Previos
LABORATORIO 4.1.3.6

Objetivos
Familiarizar al estudiante con:

- Proyectar y escribir funciones parametrizadas.
- Utilizar la instrucción return.
- Utilizar las funciones propias del estudiante.

Escenario
Tu tarea es escribir y probar una función que toma dos argumentos (un año y un mes) y devuelve el número de días del mes/año dado (mientras que solo febrero es sensible al valor year, tu función debería ser universal).

La parte inicial de la función está lista. Ahora, haz que la función devuelva None si los argumentos no tienen sentido.

Por supuesto, puedes (y debes) utilizar la función previamente escrita y probada (LABORATORIO 4.1.3.6). Puede ser muy útil. Te recomendamos que utilices una lista con los meses. Puedes crearla dentro de la función; este truco acortará significativamente el código.

Hemos preparado un código de prueba. Ampliálo para incluir más casos de prueba.

```
2 def is_year_leap(year):
3     #
4     # Tu código del LABORATORIO 4.3.6.
5     #
6     if year % 4 == 0:
7         if year % 100 == 0:
8             if year % 400 == 0:
9                 return True
10            else:
11                return False
12        else:
13            return True
14    else:
15        return False
16
17 def days_in_month(year, month):
18     # Verificar si el mes es válido
19     if month < 1 or month > 12:
20         return None
21
22     # Días de cada mes en años no bisiestos
23     days_in_months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
24
25     # Si el mes es febrero y el año es bisiesto, devolver 29 días
26     if month == 2 and is_year_leap(year):
27         return 29
28
29     # Devolver los días correspondientes al mes
30     return days_in_months[month - 1]
31
```

Console >_

1900 2 ->OK
2000 2 ->OK
2016 1 ->OK
1987 11 ->OK

4.3.1.8 LABORATORIO: Día del año: escribiendo y utilizando tus propias funciones

En este laboratorio, aprendí a crear funciones con múltiples parámetros. Desarrollé una función que toma un año, un mes y un día como argumentos, y devuelve el día correspondiente del año, o None si alguno de los argumentos no es válido.



« 4.3.1.8 LABORATORIO: Día del año: escribiendo y utilizando tus propias funciones »

LABORATORIO

Tempo Estimado
30 minutos

Nivel de Dificultad
Medio

Requisitos Previos
LABORATORIO 4.1.3.6
LABORATORIO 4.1.3.7

Objetivos
Familiarizar al estudiante con:

- Proyectar y escribir funciones con parámetros.
- Utilizar la sentencia return.
- Construir un conjunto de funciones de utilidad.
- Utilizar las funciones propias del estudiante.

Escenario
La tarea es escribir y probar una función que toma tres argumentos (un año, un mes y un día del mes) y devuelve el día correspondiente del año, o devuelve None si cualquiera de los argumentos no es válido.

Después de utilizar las funciones previamente escritas y probadas. Agrega algunos casos de prueba al código. Esta prueba es solo el comienzo.

```
1 #Juan Miguel Hernández Beltrán
2 # Función para determinar si un año es bisiesto (estilo basado en las imágenes)
3 def is_year_leap(year):
4     if year % 4 == 0:
5         if year % 100 == 0:
6             if year % 400 == 0:
7                 return True
8             return False
9         return True
10    return False
11
12 # Función para devolver los días de un mes en un año dado
13 def days_in_month(year, month):
14     # Verificar si el mes es válido
15     if month < 1 or month > 12:
16         return None
17
18     # Lista con los días de cada mes para un año no bisiesto
19     days_in_months = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
20
21     # Si el mes es febrero y el año es bisiesto
22     if month == 2 and is_year_leap(year):
23         return 29
24
25     # Devolver los días del mes correspondiente
26     return days_in_months[month - 1]
27
28 # Función para devolver el día correspondiente del año
29 def day_of_year(year, month, day):
30     # Verificar si el mes es válido
31     days_this_month = days_in_month(year, month)
32     if days_this_month is None or day < 1 or day > days_this_month:
33         return None
34
35     # Acumulador para los días
36     total_days = 0
37     for m in range(1, month):
38         total_days += days_in_month(year, m)
39
40     # Añadir el día actual
```

Console >_

366
60
60
None
319

4.3.1.9 LABORATORIO: Números primos: ¿Cómo encontrarlos?

En este laboratorio, aprendí sobre los conceptos y algoritmos clásicos relacionados con los números primos. Escribí una función llamada is_prime que toma un número como argumento y verifica si es primo.



« 4.3.1.9 LABORATORIO: Números primos: ¿Cómo encontrarlos? »

Objetivos

- Familiarizar al estudiante con nociones y algoritmos clásicos.
- Mejorar las habilidades del estudiante para definir y emplear funciones.

Escenario
Un número natural es **primo** si es mayor que 1 y no tiene divisores más que 1 y sí mismo.

¿Complicado? De ninguna manera. Por ejemplo, 8 no es un número primo, ya que puedes dividirlo entre 2 y 4 (no podemos usar divisores iguales a 1 y 8, ya que la definición lo prohíbe).

Por otra parte, 7 es un número primo, ya que no podemos encontrar ningún divisor para él.

Tu tarea es escribir una función que verifique si un número es primo o no.

La función:

- Se llama is_prime.
- Toma un argumento (el valor a verificar).
- Devuelve True si el argumento es un número primo, y False de lo contrario.

Sugerencia: intenta dividir el argumento por todos los valores posteriores (comenzando desde 2) y verifica el resto: si es cero, tu número no puede ser un número primo; analiza cuidadosamente cuándo deberías detener el proceso.

Si necesitas conocer la raíz cuadrada de cualquier valor, puedes utilizar el operador **. Recuerda: la raíz cuadrada de x es lo mismo que $x^{0.5}$.

Complementa el código en el editor.

Ejecuta tu código y verifica si tu salida es la misma que la nuestra.

Datos de prueba
Salida esperada:

1 2 3 4 5 6 7 11 13 17 19


```
1 #Juan Miguel Hernández Beltrán
2 def is_prime(num):
3     #
4     # Escribe tu código aquí.
5     #
6     if num <= 1:
7         return False
8     # Verificar si el número es divisible por algún valor entre 2 y la raíz cuadrada del número
9     for i in range(2, int(num ** 0.5) + 1):
10        if num % i == 0:
11            return False
12        return True
13
14 # Ciclo para verificar los números entre 2 y 20
15 for i in range(1, 20):
16     if is_prime(i + 1): # Verificar si i+1 es primo
17         print(i + 1, end=" ")
18
19 print() # Para añadir un salto de línea al final
```

Console >_

2 3 5 7 11 13 17 19

4.3.1.10 LAB: Convirtiendo el consumo de combustible

En este laboratorio, mejoré mis habilidades para definir, utilizar y probar funciones en Python. Escribí dos funciones: `liters_100km_to_miles_gallon`, que convierte el consumo de combustible de litros por cada 100 kilómetros a millas por galón, y `miles_gallon_to_liters_100km`, que realiza la conversión inversa.



Python INSTITUTE

« 4.3.1.10 LAB: Convirtiendo el consumo de combustible »

MODULE (36%)

SECTION (91%)

Nivel de Dificultad

ácil

Objetivos

- Mejorar las habilidades del estudiante para definir, utilizar y probar funciones.

Escenario

El consumo de combustible de un automóvil se puede expresar de muchas maneras diferentes. Por ejemplo, en Europa, se muestra como la cantidad de combustible consumido por cada 100 kilómetros.

En los EE. UU., se muestra como la cantidad de millas recorridas por un automóvil con un galón de combustible.

Tu tarea es escribir un par de funciones que conviertan l/100km a mpg (millas por galón), y viceversa.

Las funciones:

- Se llaman `liters_100km_to_miles_gallon` y `miles_gallon_to_liters_100km` respectivamente.
- Toman un argumento (el valor correspondiente a sus nombres).

Complementa el código en el editor.

Ejecuta tu código y verifica si tu salida es la misma que la nuestra.

¡Aquí hay información para ayudarte:

- 1 milla = 1609.344 metros.
- 1 galón = 3.785411784 litros.

Salida esperada:

```
60.31143162393162
31.361944444444444
23.521458333333333
3.9007393587617467
7.490910297239916
10.009131205673757
```

1 #Juan Miguel Hernández Beltrán
2 def liters_100km_to_miles_gallon(liters):
3 #
4 # Escribe tu código aquí.
5 return 235.215 / liters
6 def miles_gallon_to_liters_100km(miles):
7 #
8 # Escribe tu código aquí.
9 return 235.215 / miles
10 print(liters_100km_to_miles_gallon(3.9))
11 print(liters_100km_to_miles_gallon(7.5))
12 print(liters_100km_to_miles_gallon(10.))
13 print(miles_gallon_to_liters_100km(60.3))
14 print(miles_gallon_to_liters_100km(31.4))
15 print(miles_gallon_to_liters_100km(23.5))
16

Console >_

File "main.py", line 6
def miles_gallon_to_liters_100km(miles):
^
IndentationError: expected an indented block
60.31153846153846
31.362000000000002
23.5215
3.9007462686567167
7.4909235668789815

4.7.2.1 PROYECTO: TIC-TAC-TO

En este laboratorio desarrolle un programa que simula el juego de tic-tac-toe. Implementé un tablero utilizando una lista de listas, donde cada elemento representa una celda del juego. La máquina juega con 'X' y el usuario con 'O', comenzando siempre con una 'X' en el centro.

SECCIÓN 100%

SECCIÓN 100%

SECCIÓN 100%

tempo Estimado

>50 minutos

nivel de Dificultad

edio/Difcil

Objetivos

- Perfeccionar las habilidades del estudiante al emplear Python para resolver problemas complejos.
- La integraci3n de t3cnicas de programaci3n en un solo programa consistente de varias partes.

scenari

Tareas es escribir un simple programa que simele jugar a tic-tac-toe (nombre en ingl3s) con el usuario. Para hacerlo m3s f3cil, vnos decidid simplificar el juego. Aqu3 est3n nuestras reglas:

- La maquina (por ejemplo, el programa) jugar3 utilizando las 'X' s.
- El usuario (por ejemplo, tu) jugar3 utilizando las 'O' s.
- El primer movimiento es de la maquina: siempre coloca una 'X' en el centro del tablero.
- Todos los cuadros est3n numerados comenzando con el 1. (Observa el ejemplo para que tengas una referencia).
- El usuario ingresa su movimiento introduciendo el n3mero de cuadro elegido. El n3mero debe de ser valido, por ejemplo un valor entero mayor que 0 y menor que 10, y no puede ser un cuadro que ya est3 ocupado.
- El programa verifica si el juego ha terminado. Existen cuatro posibles veredictos: el juego continua, el juego termina en empate, tu ganas, o la maquina gana.
- La maquina responde con su movimiento y se verifica el estado del juego.
- No se debe implementar alg3n tipo de inteligencia artificial, la maquina elegir3 un cuadro de manera aleatoria, eso es suficiente para este juego.

ejemplo del programa es el siguiente:

1	2	3
4	X	6
7	8	9

Ingresa tu movimiento: 1

0	2	3
---	---	---

```
1 from random import randint
2
3 # Funci3n para mostrar el tablero en la consola
4 def DisplayBoard(board):
5     for row in board:
6         print("-----" + 3 * "+" + "-----")
7         for cell in row:
8             print("|" + str(cell) + " |", end="")
9         print("\n")
10    print("-----" + 3 * "+" + "-----")
11
12 # Funci3n para que el usuario ingrese su movimiento
13 def EnterMove(board):
14     while True:
15         move = input("Ingresa tu movimiento (1-9): ")
16         if move.isdigit():
17             move = int(move)
18             if move >= 1 and move <= 9:
19                 for row in range(3):
20                     for col in range(3):
21                         if board[row][col] == move:
22                             if board[row][col] == 'O':
23                                 board[row][col] = 'X'
24                             return
25                 print("Movimiento inv3lido. Int3ntalo de nuevo.")
26
27 # Funci3n para encontrar los cuadros vacios
28 def MakeListOfFreeFields(board):
29     free = []
30     for row in range(3):
31         for col in range(3):
32             if isinstance(board[row][col], int): # Si es un n3mero, est3 vacio
33                 free.append((row, col))
34     return free
35
36 # Funci3n para verificar si hay un ganador
37 def VictoryFor(board, sign):
38     for row in range(3):
39         if all([cell == sign for cell in board[row]]): # Comprobar filas
40             return True
41     for col in range(3):
```

```
=====
| 1 | 2 | 3 |
+-----+
| 4 | X | 6 |
+-----+
| 7 | 8 | 9 |
+-----+
Ingresa tu movimiento (1-9): 9
=====
| 1 | 2 | 3 |
+-----+
| 4 | X | X |
+-----+
| 7 | 8 | 0 |
+-----+
Ingresa tu movimiento (1-9): 4
=====
| 1 | 2 | 3 |
+-----+
| 4 | X | X |
+-----+
| 7 | 8 | 0 |
+-----+
```

Prev

Next

Pantalla de Quiz

Python INSTITUTE

PE1 - Module 4 Quiz

Progress (100%)

Your score: 10/12

83%

Congratulations, you've passed the quiz!

SECTION ANALYSIS

PE1 - Module 4 Quiz 83%

Retake Test

Review Test

Pantalla de Examen Módulo 4

Progress (100%)

Your score: 16/22

73%

Congratulations, you've passed the test!

SECTION ANALYSIS

PE1 -- Module 4 Test73%

Retake Test

Review Test