PythonNotebook3_solution_2023

December 4, 2023

Exercise 3.1.1

In this exercise you will write your own Celsius to Fahrenheit converter! Your task is to write a function, which will accept the list of temperatures temp_c, in Celsius, and will output a list with the same temperatures, but in Fahrenheit.

Hint: create an empty list for the result, then append values to this list. See the "personalized greeting" example above.

[ ]:
```
# you do not need to change anything in this cell

temperatures_c = [-1, -1.2, 1.3, 6.4, 11.2, 14.8, 17.8, 17.7, 13.7, 8.5, 4.1, 0.
↪9]
```

[ ]:
```
def celsius_to_fahrenheit(temp_c):
    ...

###BEGIN SOLUTION TEMPLATE=
def celsius_to_fahrenheit(temp_c):
    temp_f = []

    for i in range(len(temp_c)):
        temp_f.append(temp_c[i] * 9 / 5 + 32)
    return temp_f
###END SOLUTION

print(celsius_to_fahrenheit(temp_c))
```

Exercise 3.1.2

Your task here is to write a function which will analyze a broadcasting message of the following format: "satellite_ids;date", where the first part of the message contains unique lowercase letters, each corresponding to a different satellite ID, and the last part contains the date of the message. Here are some examples: "agf;06062022" (3 satellites: a, g, and f), "abcdefgops;03121999" (10 satellites), "xyz;11112011" (3 satellites). Your task is to write a function, which for a provided broadcast message, will count the number of satellites mentioned in the message.

[ ]:
```
def count_satellites(message):
    ...
```

```
###BEGIN SOLUTION TEMPLATE=
def count_satellites(message):
    for i in range(len(message)):
        if message[i] == ';':
            break

    return i
###END SOLUTION

# Check that with the example below you count 5 satellites
print(count_satellites("hallo;12122007"))
```

Exercise 3.1.3

Here you need to write a function that is able to sort any list consisting only of real numbers, in the descending order. For example, the list $[19, 5, 144, 6]$ becomes $[144, 19, 6, 5]$. Hint: use a built-in sort() function to sort the list in ascending order, and then think of a clever way to change the order this list to descending order.

[ ]:
```
def sort_list(unsorted_list):
    ...

###BEGIN SOLUTION TEMPLATE=
def sort_list(unsorted_list):
    return sorted(unsorted_list)[::-1]
###END SOLUTION


print(sort_list([9, -1, 5, 1, -9, -9]))
```

Exercise 3.1.4

Use a for loop and what you learned above about range() to print out every third element of the list L, starting from the second.

[2]:
```
L = ['a', '1', 'b', 'c', '2', 'd', 'e', '3', 'f', 'g', '4', 'h'] # you don't␣
 ↪need to change L

# your loop here
...

###BEGIN SOLUTION TEMPLATE=

# OPTION 1: list comprehension

every_third_element_L = [L[i] for i in range(len(L)) if (i+1)%3 == 0]

# OPTION 2: for loop
```

```
every_third_element_L = []

for i in range(len(L)):
    if (i+1)%3 == 0:
        every_third_element_L.append(L[i])

# OPTION 3: using the step in range()

every_third_element_L = []

for i in range(2, len(L), 3):
    every_third_element_L.append(L[i])

print(every_third_element_L)


###END SOLUTION
```

```
['b', 'd', 'f', 'h']
```

3.2 More functions and loops practice

Exercise 3.2.1

Write a function that converts a number from degrees to radians.

[ ]:
```
def DegToRad(deg):
    ...

###BEGIN SOLUTION TEMPLATE=

def DegToRad(deg):
    """
    Convert degrees to radians.

    Parameters:
     deg (float): Angle in degrees.

    Returns:
     float: Angle in radians.
    """

    return deg * (3.1415 / 180)

###END SOLUTION

Angle = 180 # Degrees
print(f"An angle of {Angle} Degrees is equal to {DegToRad(180):.3f} radians")
```

Exercise 3.2.2

Write a function that takes four inputs: (x_1, y_1, x_2, y_2) and computes the Euclidian distance between point 1 (x_1, y_1) and point 2 (x_2, y_2).

[ ]:
```
import math #

# def distance ...

###BEGIN SOLUTION TEMPLATE=
def distance(x1,y1,x2,y2):
    """
    Calculate the Euclidean distance between two points (x1, y1) and (x2, y2).

    Parameters:
     x1 (float): x-coordinate of the first point.
     y1 (float): y-coordinate of the first point.
     x2 (float): x-coordinate of the second point.
     y2 (float): y-coordinate of the second point.

    Returns:
     float: Euclidean distance between the two points.
    """

    # OPTION 1: without using the `math` library

    return ((x2 - x1)**2 + (y2 - y1)**2)**0.5

    # OPTION 2: with the `math` library

    return math.dist((x1,y1),(x2,y2))
###END SOLUTION

x1, y1 = 1, 1
x2, y2 = 2, 3

print(f"Distance between points ({x1}, {y1}) and ({x2}, {y2}) is {distance(x1,␣
 ↪y1, x2, y2):.3f}")
```

Exercise 3.2.3

Write a function that determines if its argument is a prime number.

- Hint 1: an integer n is prime if n > 1 and n is not divisible by any smaller integer > 1
- Hint 2: divisibility can be tested with the % operator. If n % i is 0, n is divisible by i.

[ ]:
```
def is_prime(n):
    return True # or False

###BEGIN SOLUTION TEMPLATE=
def is_prime(n):
```

```
    """
    Determine if a number is a prime number.

    Parameters:
     n (int): The number to check.

    Returns:
     bool: True if n is prime, False otherwise.
    """
    if n > 1:
        for i in range(2, int(n/2)+1):
            if (n % i) == 0:
                return False
        return True
    return False
###END SOLUTION
```

Exercise 3.2.4

Use your is_prime() function to create a list of all primes < 1000. What is the sum of all primes < 1000?

Hint: is there a nice way to calculate the sum of the elements in a list?

[ ]:
```
prime_list = ...
prime_sum = ...
###BEGIN SOLUTION TEMPLATE=

# OPTION 1: list comprehension
prime_list = [i for i in range(1000) if is_prime(i)]

# OPTION 2: for loop
prime_list = []

for i in range(1000):
    if is_prime(i): # <= same as doing `if is_prime(i) == True:`
        prime_list.append(i)

prime_sum = sum(prime_list)

###END SOLUTION
print(prime_list)
print(prime_sum)
```

(Fixing) Exercise 3.3.1

Fix the syntax errors so it prints "AES" without removing the variable that holds it. You'll need to fix 2 errors.

[ ]:
```
def get_abbreviation():
    my abbreviation = "AES"
     return my_abbreviation

###BEGIN SOLUTION TEMPLATE=
def get_abbreviation():
    my_abbreviation = "AES"
    return my_abbreviation
###END SOLUTION

print(get_abbreviation())
```

(Fixing) Exercise 3.3.2

Find the semantic error in this function.

Hint: The factorial n! is defined as n! = n * (n - 1) * (n - 2) * … * 2 * 1. The function uses the fact that if n > 0, n! = n * (n - 1)!. This is an example of a *recursive* function, a function that calls itself.

[ ]:
```
def factorial(x):
    "returns the factorial of x"
    if x == 0:
        return 1
    else:
        return x ** factorial(x-1)

###BEGIN SOLUTION TEMPLATE=
def factorial(x):
    "returns the factorial of x"
    if x == 0:
        return 1
    else:
        return x * factorial(x-1)
###END SOLUTION

factorial(4)
```