04-Exercises

November 6, 2024

1 03 - Exercises: functions, debugging, functional programming and lazy evaluation

This week we saw a looot of things: - Functions - Debugging - Functional programming - Lazy evaluation

Here are some exercises to help you get comfortable with these concepts :)

1.0.1 1. Animal Parade! (but turn this into a function) - 2 points

Imagine you're organizing a parade with a list of animals. For each animal in the list, print its name multiple times, creating a "parade" effect based on the animal's length (number of characters). For instance, "Cat" (3 letters) will repeat 3 times.

Let's break down the task:

1. Create a list of animal names (e.g., `["Cat", "Elephant", "Dog", "Giraffe", "Bee"]`).
2. Create a **function** that for each animal prints its name as many times as it has letters.
3. Define a single function and use nested for loops to print the repeated names.

Sample Output:

```
Cat Cat Cat
Elephant Elephant Elephant Elephant Elephant Elephant Elephant Elephant
Dog Dog Dog
Giraffe Giraffe Giraffe Giraffe Giraffe Giraffe Giraffe
Bee Bee Bee
```

Choose 5 different animals this time :)

[ ]:

1.0.2 2. Simple Calculator - 2 points

You decide that you want to substitute your physical calculator with a Python-based one. Write a function that performs this task!

Let's break down the task:

1. Define a function `calculator(a, b, operation)` where operation can be "add", "subtract", "multiply", or "divide".
2. Based on `operation`, return the result of the chosen operation.

For example, `calculator(10, 5, "add")` should return `15`.

[ ]:

1.0.3 3. Factorial Calculator - 2 points

Write a **recursive function** that calculates the factorial of a number. The factorial of a number `n`, denoted by `n!`, is the product of all positive integers less than or equal to `n`. E.g. `3! = 3 * 2 * 1 = 6`.

Let's break down the task:

1. Write a recursive function `factorial(n)` that returns the factorial of a given number `n`.

For example, `factorial(5)` should return `120`.

[ ]:

1.0.4 4. Food Servings Adjuster - 2 points

Write a function that uses `try/except` statements for error handling in division.

Let's break down the task:

1. Write a function `adjust_recipe(servings, people)` that calculates how much to multiply each ingredient to serve the desired number of people.
2. If people is zero (meaning there's no one to serve!), catch the error and return `"Sorry, no one to cook for!"`.

For example, `adjust_recipe(4, 2)` should return `0.5` (since you're cooking for half as many people).

Example:

```
adjust_recipe(4, 2)  # Should return 0.5
adjust_recipe(4, 8)  # Should return 2.0
adjust_recipe(4, 0)  # Should return "Sorry, no one to cook for!"
```

[ ]:

1.0.5 5. Prime Number Finder for Space Missions - 3 points

Write a function that uses the `filter` function to identify only "prime" rocket numbers.

Let's break down the task:

1. Write a function `is_prime(n)` that checks if `n` is a prime number.
2. Define a list of rocket numbers, e.g., `[101, 202, 303, 404, 505, 607, 808]`.
3. Use `filter` with `is_prime` to identify only the prime rocket numbers, meaning they're more likely to "reach space" safely!

Expected Output:

```
Promising rockets are: [101, 607]
```

[ ]:

1.0.6 6. Spot the error in the String Reverser - 3 points

Carefully read the following code and identify the errors in the string manipulation.

The `reverse_string` function is intended to return the reverse of a given string, but it's not working as expected. What are the problems? Write the correct function:

[ ]:
```
def reverse_string(s):
    reversed_str = ""
    for i in range(len(s)):
        reversed_str = reversed_str + s[i]
    return reversed

print(reverse_string("hello"))  # Expected output: "olleh"
```

[ ]:

1.0.7 7. Spot the error in the Number Doubler with `map` - 3 points

Carefully read the following code and identify the errors in the use of the `map` function.

The function is intended to return the double of each number in a list, but it's not working as expected. What are the problems? Write the correct function:

[ ]:
```
numbers = [1, 2, 3, 4]
doubled_numbers = map(lambda x: x * 2, numbers)
print(doubled_numbers)  # Expected output: [2, 4, 6, 8]
```

[ ]:

1.0.8 8. Fix the Sum with Reduce - 3 points

Carefully read the following code and identify the errors in the use of the `reduce` function.

The function is intended to return the sum of all numbers in a list, but it's not working as expected. What are the problems? Write the correct function:

[ ]:
```
from functools import reduce
numbers = [1, 2, 3, 4]
total_sum = reduce(lambda x, y: x + y)
print(total_sum)  # Expected output: 10
```

1.1 Bonus exercises

1.1.1 1. Dictionary Merger with Duplicate Key Handling - 2 points

Write a functino to merge two dictionaries with custom handling for duplicate keys.

Let's break down the task: 1. Define a function `merge_dicts` that takes two dictionaries and returns a new dictionary by merging them. 2. Use a `try/except` block to handle **duplicate** keys,

which should add `"Error: Duplicate key encountered!"` for that key. 3. What happens if you do not handle duplicate keys? What is the default Python behaviour?

Example:

```
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 3, "c": 4}
print(merge_dicts(dict1, dict2)) # Expected output: {"a": 1, "b": "Error:
Duplicate key encountered!", "c": 4}
```

[ ]: