Published using Google Docs

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

Guide

New Scratch SFT Coding Examples Standard and Advanced Coding

Standard Examples

Code Generation Category

Test Generation Category

*****Advanced Examples

RAG Category

Bug Fixing Category

Solution Reasoning Category

Test Generation Category

✓ Good Examples

□Bad Prompt Examples



Code Generation Category

Learn more

(i)

Published using Google Docs Report abuse

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

be used as stipulated, and for each unit, the delivery time is increased by 20%. However, when it reaches 5, the delivery is not made, and the object remains in the warehouse of origin until the next day. The cost of the shipment is per kilometer traveled, plus a fixed shipping cost, and there is a bonus that is paid if the object must stay in the warehouse for one more day. I am in charge of calculating the daily costs of the loads to be transported. I would need a program in Python that allows me to upload the file, enter the necessary values, and then display the costs for the day, along with a total. I also want it to have default values (\$10, \$116, and \$72 respectively). My task also consists of verifying that the file has correct data and is not incorrectly loaded or empty, so I should have a menu that allows me to choose the option to verify the quality of the file before starting the task.

_-----

Since it is a program with a certain degree of complexity and varied data manipulation, it is convenient to make a simple approach plan. The following is a plan based on dividing the tasks into basic development blocks as required:

- a) **Load the CSV data and verify its integrity:** It is important to load the file into another more flexible and manipulable data structure such as a DataFrame, performing a basic review of the content beforehand.
- b) **Verify the data:** Since it is one of the user's tasks, this process must be considered as a separate unit, checking if the DataFrame is empty or does not contain the columns necessary for the development of the calculations.
- c) **Calculate the transport cost for each shipment based on the restrictions provided:** Create a function that allows calculations to be made according to weather and route conditions, specific values for distances, and fixed costs according to the circumstances.
- d) **Calculate the costs of the day:** This is the core of the task. Group the costs of all trips into a manipulable structure, like another DataFrame, to present them on the screen.
- e) **Options menu:** Depending on the requirements, it must be possible to choose between directly developing the calculations or verifying the data before starting. Therefore, a function that appropriately manipulates an options menu is necessary.
- f) **Integrate the functions:** Finally, once a function is developed for each of the components of the problem, a code will be created to integrate all the functionalities

Report abuse

Learn more

New Scratch SFT Coding Examples

ппроп рапиаѕ аѕ ри

Updated automatically every 5 minutes

```
def compute transport cost(distance, condition, fixed cost, cost per km,
delay_bonus):
  Calculate the transport cost based on the fixed cost price by km, and delay
bonus.
  :param distance: Distance between warehouses (in kilometers).
  :param condition: Weather/route conditions (value from 1 to 5).
  :param fixed cost: Fixed cost for shipping.
  :param cost per km: Price per kilometer.
  :param delay_bonus: Bonus applied if the shipment is delayed for condition 5.
  :return: Total transport cost.
  if condition == 5:
     # Case when a shipment is delayed, then apply a bonus
     return fixed_cost + delay bonus
  else:
     # Case when the shipment is in condition to travel, adjust the time based on
the weather or route conditions
     adjusted_time_multiplier = 1 + (0.2 * (condition - 1))
     return fixed cost + (distance * cost per km * adjusted time multiplier)
def import_csv_data(file_path):
  Imports data from the CSV file and returns it as a pandas DataFrame.
  :param file path: Path to the CSV file.
  :return: DataFrame containing the CSV data.
  try:
     return pd.read csv(file path)
  except FileNotFoundError:
     print(f"Error: The file '{file_path}' was not found.")
     return None
  except pd.errors.EmptyDataError:
     print(f"Error: The file '{file path}' is empty.")
     return None
  except pd.errors.ParserError:
```

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
distances.
  :param dataframe: DataFrame containing the shipment data.
  :param fixed cost: User-provided fixed cost for shipping.
  :param cost per km: User-provided cost per kilometer.
  :param delay bonus: User-provided bonus for delayed shipments.
  :return: Tuple containing DataFrame with the calculated costs for each shipment
and the total cost.
  transport costs = []
    for index, row in dataframe.iterrows():
       cost = compute transport cost(row['Distance'], row['Condition'], fixed cost,
cost per km, delay bonus)
       transport costs.append({
         'ID': row['ID'],
         'Origin Warehouse': row['Origin'],
          'Destination Warehouse': row['Destination'],
         'Distance': row['Distance'],
          'Condition': row['Condition'],
          'Cost': cost
       })
    # Transform the list of costs to a DataFrame
    cost_df = pd.DataFrame(transport_costs)
    # Calculate total cost
    total cost = cost df['Cost'].sum()
    return cost_df, total_cost
  except KeyError as e:
    print(f"Error: Missing column in the CSV file: {e}")
    return None, 0
def get user input(prompt, default value=None):
  Get user input and handle invalid inputs.
```

Calculate the daily transport costs for each shipment based on conditions and

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
value = input(prompt)
       if value == " and default value is not None:
          return default value
       return float(value)
     except ValueError:
       print("Invalid input. Please enter a valid number.")
def verify_file_quality(dataframe):
  Verify whether the CSV file has the correct structure and is not empty.
  :param dataframe: The loaded DataFrame from the CSV file.
  :return: True if the file is valid, False otherwise.
  required_columns = ['ID', 'Origin', 'Destination', 'Distance', 'Condition']
  if dataframe is None:
     return False
  # Verify if the DataFrame is empty
  if dataframe.empty:
    print("Error: The CSV file is empty.")
     return False
  # Verify if all required columns are present
  for column in required columns:
     if column not in dataframe.columns:
       print(f"Error: The required column '{column}' is missing.")
       return False
  print("The CSV file is valid.")
  return True
def display_menu():
  Shows a menu to the user and prompts for a choice.
  :return: The choice made by the user.
  print("\nMenu:")
  print("1. Verify CSV file quality")
```

Learn more

Published using Google Docs

Report abuse

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
# Path to the CSV file
  file path = 'shipments data day1.csv'
  df = import csv data(file path)
  while True:
     choice = display menu()
     if choice == '1':
       # Verify file quality
       valid_file = verify_file_quality(df)
     elif choice == '2':
       # Verify if file is valid before proceeding
       if verify_file_quality(df):
          # Request to the user to input values for calculations
          fixed cost = get user input("Enter the fixed cost for shipping (default
116): ", default value=116)
          cost per km = get user input("Enter the cost per kilometer (default 10):
", default value=10)
          delay bonus = get user input("Enter the delay bonus for shipments with
condition 5 (default 72): ", default value=72)
          # Calculate the daily costs if the DataFrame is valid
          result df, total cost = compute daily transport costs(df, fixed cost,
cost per km, delay bonus)
          if result df is not None:
            # Display the DataFrame with calculated costs and the total cost
            print("\n=== Shipment Cost Details ===\n")
            print(result df)
            print(f"\nTotal cost for all shipments: {total cost}")
        else:
          print("Please verify the CSV file first.")
     elif choice == '3':
        print("Exiting the program.")
        break
     else:
        print("Invalid choice. Please choose a valid option.")
```

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

Test Generation Category

Prompt: I am a teaching assistant in the numerical methods department at my faculty, and we are currently developing code combining symbolic and numerical calculus. At the moment we have the following code for the Taylor series, in a file called `tylor.py`:

```
```Python
import numpy as np
import sympy as sym
def taylor_poly(fx, x0, n):
 k = 0
 polynomial = 0
 while k \le n:
 derivative = fx.diff(x, k)
 derivative at x0 = derivative.subs(x, x0)
 divisor = np.math.factorial(k)
 term k = (derivative at x0 / divisor) * (x - x0) ** k
 polynomial = polynomial + term k
 k = k + 1
 return polynomial
def manipulate function(fx, scale factor=1, shift value=0):
 manipulated_fx = scale_factor * fx.subs(x, x + shift_value)
 return manipulated_fx
def combine functions(fx, other fx, operation='add'):
 if operation == 'add':
 return fx + other_fx
 elif operation == 'multiply':
 return fx * other fx
 elif operation == 'subtract':
 return fx - other fx
 elif operation == 'divide':
```

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
def lagrange remainder(fx, x0, xi, n):
 derivative n plus 1 = fx.diff(x, n+1)
 derivative_at_xi = derivative_n_plus_1.subs(x, xi)
 remainder = (derivative at xi / np.math.factorial(n+1)) * (xi - x0)**(n+1)
 return remainder
def relative_error(fx, xi, approx):
 real_value = fx.subs(x, xi)
 return abs(real value - approx) / abs(real value)
x = sym.Symbol('x')
fx = sym.sqrt(x + 1)
x0 = 0
xi = 0.5
n = 3
fx manipulated = manipulate function(fx, scale factor=2, shift value=-0.5)
trig function = sym.sin(x)
combined_fx = combine_functions(fx_manipulated, trig_function, operation='add')
fxi = combined fx.subs(x, xi)
polynomial = taylor_poly(combined_fx, x0, n)
pxi = polynomial.subs(x, xi)
real error = fxi - pxi
relative err = relative error(combined fx, xi, pxi)
error_bound = lagrange_remainder(combined_fx, x0, xi, n)
print('Original Function: ', fx)
print('Manipulated Function: ', fx manipulated)
print('Combined Function: ', combined_fx)
print('Taylor Polynomial:
 ', polynomial)
print('xi:
 ', xi)
print('Estimated (Taylor): ', pxi)
print('Real:
 ', fxi)
print('Real Error:
 ', real error)
print('Relative Error: ', relative err)
print(f'Error Bound (Degree {n}):', error_bound)
```

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

Here is the requested code with multiple examples covering several of the most important edge cases:

""Python
import unittest

import sympy as sym from termcolor import colored from taylor import taylor\_poly, combine\_functions, lagrange\_remainder, manipulate\_function, relative\_error

 $class\ Colored Text Test Result (unit test. Text Test Result):$ 

"""Custom TestResult class to print colored text for unittest results."""

def addSuccess(self, test):
 super().addSuccess(test)
 print(colored(f"OK: {test}", "green")) # Display success in green

def addFailure(self, test, err):
 super().addFailure(test, err)
 print(colored(f"FAIL: {test}", "red")) # Display failure in red

def addError(self, test, err):
 super().addError(test, err)
 print(colored(f"ERROR: {test}", "red")) # Display errors in red

class ColoredTextTestRunner(unittest.TextTestRunner):
 """Custom TestRunner to use ColoredTextTestResult."""

def \_makeResult(self):
 return ColoredTextTestResult(self.stream, self.descriptions, self.verbosity)

class TestTaylorSeries(unittest.TestCase):

```
def setUp(self):
 """Set up common variables for the tests."""
 self.x = sym.Symbol('x')
 self.fx = sym.sqrt(self.x + 1)
 self.trig_function = sym.sin(self.x)

def test taylor poly basic(self):
```

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
value should match the function at x0.")
 def test taylor poly degree zero(self):
 """Test edge case when n = 0 (Taylor series of degree 0)."""
 n = 0 # Degree 0 Taylor polynomial (constant)
 poly = taylor poly(self.fx, x0, n)
 expected = self.fx.subs(self.x, x0)
 self.assertEqual(poly, expected, "Degree 0 Taylor polynomial should equal the
function value at x0.")
 def test manipulate function scale and shift(self):
 """Test the function manipulation with scaling and shifting."""
 scale factor = 3
 shift_value = -0.6 # Applying a shift and scale to the function
 manipulated_fx = manipulate_function(self.fx, scale_factor, shift_value)
 expected fx = 3 * sym.sgrt(self.x - 0.6 + 1)
 self.assertEqual(manipulated fx, expected fx, "Function should be correctly
scaled and shifted.")
 def test combine functions add(self):
 """Test combining functions by addition."""
 combined fx = combine functions(self.fx, self.trig_function, operation='add')
 expected fx = self.fx + sym.sin(self.x)
 self.assertEqual(combined fx, expected fx, "Expected the sum of the two
functions.")
 def test combine functions multiply(self):
 """Test combining functions by multiplication."""
 combined fx = combine functions(self.fx, self.trig function,
operation='multiply')
 expected fx = self.fx * sym.sin(self.x)
 self.assertEqual(combined fx, expected fx, "Expected the product of the two
functions.")
 def test combine functions invalid operation(self):
 """Test invalid operation handling."""
 with self.assertRaises(ValueError, msg="Invalid operation should raise a
ValueError."):
 combine functions(self.fx, self.trig function, operation='invalid')
 def test_lagrange_remainder(self):
 """Test Lagrange remainder calculation."""
```

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
remainder should be greater than zero.")
 def test relative error(self):
 """Test relative error calculation."""
 approx = 1.2 # Approximation provided for comparison
 relative err = relative error(self.fx, xi, approx)
 self.assertGreaterEqual(relative err, 0, "Relative error should be non-
negative.")
 self.assertLess(relative err, 1, "Relative error should be less than 1 for close
approximations.")
 def test large n for taylor poly(self):
 """Test edge case for a large n (degree of Taylor polynomial)."""
 x0 = 0
 n = 25 # Large degree Taylor polynomial
 poly = taylor poly(self.fx, x0, n)
 self.assertIsInstance(poly, sym.Expr, "Expected a symbolic expression for the
Taylor polynomial even for large n.")
 def test combination with zero function(self):
 """Test combining with zero function."""
 zero function = 0
 combined fx = combine functions(self.fx, zero function, operation='add') #
Adding zero
 self.assertEqual(combined fx, self.fx, "Adding zero should return the original
function.")
 combined fx multiply = combine functions(self.fx, zero function,
operation='multiply') # Multiplying by zero
 self.assertEqual(combined fx multiply, 0, "Multiplying by zero should return
zero.")
 def test negative x(self):
 """Test with negative values for x."""
 x0 = -0.45
 xi = -0.55
 poly = taylor_poly(self.fx, x0, 3) # Using a 3rd-degree Taylor polynomial
 result = poly.subs(self.x, xi)
 real value = self.fx.subs(self.x, xi)
 self.assertAlmostEqual(result, real value, delta=1e-3, msg="Taylor polynomial
should approximate the function for negative x.")
```

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
should give a good approximation for large x values.")
 def test xi equals x0(self):
 """Test when xi is the same as x0."""
 x0 = 2.0
 xi = 2.0 # When xi equals x0, Taylor polynomial should match the function
exactly
 poly = taylor poly(self.fx, x0, 3)
 result = poly.subs(self.x, xi)
 real value = self.fx.subs(self.x, xi)
 self.assertEqual(result, real_value, "Taylor polynomial should match the
function exactly when xi equals x0.")
 def test invalid xi(self):
 """Test with invalid (non-numeric) xi."""
 with self.assertRaises(sym.SympifyError, msg="Expected SympifyError when
passing invalid xi (None)."):
 xi = None # Passing None as an invalid input
 poly = taylor poly(self.fx, 0, 3)
 poly.subs(self.x, xi)
 def test division by zero(self):
 """Test combining functions with division by zero."""
 with self.assertRaises(ZeroDivisionError, msg="Expected ZeroDivisionError
when dividing by zero."):
 combine functions(self.fx, 0, operation='divide') # Dividing by zero should
raise ZeroDivisionError
if name == ' main ':
 unittest.main(testRunner=ColoredTextTestRunner())
We have several categories of test cases covering various topics for a class related
to the software provided and unit tests:
Some **Test Validation** confirms the fundamental functionality of Taylor
polynomial computations by comparing them with expected outcomes in various
```

self.assertAlmostEqual(result, real\_value, delta=1e-1, msg="Taylor polynomial

12 of 45

scenarios. The `test\_taylor\_poly\_basic` validates the accurate evaluation of a fourth-degree Taylor polynomial at `x0` whereas `test\_taylor\_poly\_degree\_zero` verifies the output when the polynomial's degree is zero, representing the function's

value at 'x0'.

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

verifies the Taylor approximation with very large `x` values, while `test\_xi\_equals\_x0` confirms the polynomial's accuracy when `xi` equals `x0`. Additionally, `test\_large\_n\_for\_taylor\_poly` checks the function's capability with high Taylor polynomial degrees, and `test\_negative\_x` assesses behavior with negative `x` values.

There are also \*\*Specific Tests for Error Conditions\*\*. The `test\_invalid\_xi` function ensures that passing invalid inputs, such as `None`, raises the correct exception, while `test\_division\_by\_zero` validates that dividing by zero during function combination triggers a `ZeroDivisionError`. Tests like `test\_lagrange\_remainder` and `test\_relative\_error` verify the calculation of remainder terms and relative error.

\_\_\_\_\_

# \*Advanced Examples

#### **RAG Category**

\_\_\_\_\_

**Prompt:** I am teaching the first steps in simulations. Now I want you to use the example with enumerations to simulate a compass on this webpage <a href="https://docs.swift.org/swift-book/documentation/the-swift-programming-language/classesandstructures/">https://docs.swift.org/swift-book/documentation/the-swift-programming-language/classesandstructures/</a>. With this example, you must build a program that works on any platform. It must be console-based, allowing the user to enter a direction, as well as the option to choose a random direction. Also, when the user leaves the application, the last location is given. Just like the example on the web page, show the total number of selections, as well as the number of times each direction was chosen and an associated probability, with two decimal places.

\_\_\_\_\_\_

#### Response:

Here is the app that simulates a compass with the required specifications:

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
case west = "West"
 // Methods to change the direction
 mutating func turnNorth() {
 self = .north
 mutating func turnSouth() {
 self = .south
 mutating func turnEast() {
 self = .east
 mutating func turnWest() {
 self = .west
 // Method to change the direction based on input
 mutating func turn(direction: String) {
 switch direction.lowercased() {
 case "north":
 self.turnNorth()
 case "south":
 self.turnSouth()
 case "east":
 self.turnEast()
 case "west":
 self.turnWest()
 default:
 print("Invalid direction. Please enter North, South, East, or West.")
// Initialize the current direction and remember the initial direction
var currentDirection = CompassPoint.west
let rememberedDirection = currentDirection
// Dictionary to count the number of times each direction is selected
var directionCount: [CompassPoint: Int] = [.north: 0, .south: 0, .east: 0, .west: 0]
```

case east = "East"

Published using Google Docs

Report abuse

Learn more

#### New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
// Function to read user input
func readInput() -> String? {
 return readLine()
// Function to get a random direction
func getRandomDirection() -> CompassPoint {
 return CompassPoint.allCases.randomElement()!
// Main loop to handle user input
while true {
 promptUser()
 if let input = readInput() {
 if input.lowercased() == "exit" {
 break // Exit the loop if user types 'exit'
 } else if input.lowercased() == "random" {
 currentDirection = getRandomDirection()
 directionCount[currentDirection, default: 0] += 1
 print("Random direction selected: \(currentDirection.rawValue)")
 } else {
 currentDirection.turn(direction: input)
 if let direction = CompassPoint(rawValue: currentDirection.rawValue) {
 directionCount[direction, default: 0] += 1
// Calculate the total number of selections
let totalSelections = directionCount.values.reduce(0, +)
print("\nTotal number of selections: \(totalSelections)")
// Print direction selection details
print("\nDirection Selection Details:")
for (direction, count) in directionCount {
 let probability = totalSelections > 0 ? (Double(count) / Double(totalSelections) *
100): 0.0
 print("\(direction.rawValue): \(count) selections, Probability: \(String(format:
"%.2f", probability))%")
```

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

provided. It initializes current and remembered directions, tracks direction counts, prompts the user for input with the `promptUser` function, reads user input with the `readInput` function, and gets a random direction with the `getRandomDirection` function. Processes user input in a loop calculates and prints the total selections and probabilities, and displays the final and remembered directions.

\_\_-----

#### **Bug Fixing Category**

\_\_\_\_\_

**Prompt:** I'm working at a sports statistics company. My colleague is a Python specialist. I'm unfamiliar with this language, and he created an application for conducting basic statistical calculations on NBA teams from the last season. As my colleague is currently traveling and out of reach, my boss requested that I verify if the test code is complete. Could you assist me by checking the code for bugs or any potential errors? This code must run in all environments. This is the code:

```Python import time import pandas as pd import requests from scipy.stats import kurtosis from nba api.stats.endpoints import TeamInfoCommon from nba api.stats.static import teams from requests.adapters import HTTPAdapter from requests.packages.urllib3.util.retry import Retry from requests.exceptions import ReadTimeout, RequestException, ConnectionError from urllib3.exceptions import ProtocolError from http.client import RemoteDisconnected session = requests.Session() retry strategy = Retry( total=10. backoff factor=0.3, status forcelist=[429, 500, 502, 503, 504],

allowed methods=["HEAD", "GET", "OPTIONS"]

Learn more

Published using Google Docs

Report abuse

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
uer senu_api_requesi(seii, enupoini, params).
       return session.get(endpoint, params=params, verify=True, timeout=90)
    except (ReadTimeout, RequestException, RemoteDisconnected,
ProtocolError, ConnectionError) as e:
       print("Request failed for URL:" + endpoint " with error: " + e)
       return None
TeamInfoCommon.nba stats http = CustomNBAStatsHTPP()
nba teams = teams.get teams()
team_ids_dict = {team['full_name']: team['id'] for team in nba_teams}
df = pd.DataFrame()
def fetch_team_info(team_name, team_id, retries=3):
  for attempt in range(retries):
       team info = TeamInfoCommon(team name=team id)
       if team info:
         return team_info.get_data_frames()[0]
    except (ReadTimeout, RequestException, RemoteDisconnected,
ProtocolError, ConnectionError) as e:
       print(f"Attempt { attempt + 1} failed for team {team name}: {e}")
       time.sleep(0.01)
  print(f"Failed to fetch data for team {team_name} after {retries} attempts.")
  return None
for team name, team id in team ids dict.items():
  df team = fetch team info(team name, team id)
  if df team is not None:
    df team['Season'] = '2023-24'
    df = pd.concat([df, df team], ignore index=True)
  else:
    print(f"Skipping team {team name} due to repeated failures.")
if 'Wins' in df.columns and 'Losses' in df.columns and 'PCT' in df.columns:
  df['GP'] = df['Wins'] + df['Losses']
  df['WinPercentage'] = df['PCT'] * 100
  df['Rank'] = df['WinPercentage'].rank(ascending=False)
```

Published using Google Docs

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
"Median Wins": median_wins,
   "Variance Wins": variance_wins,
   "Standard Deviation Wins": std_dev_wins,
   "Kurtosis Wins": kurtosis_wins
}

print("Overall Team Statistics for Wins (rounded to 2 decimals):")
for stat, value in team_stats.items():
   print(f"{stat}: {value}")

print("\nTeam Statistics Summary:")
df['WinPercentage'] = df['WinPercentage'].round(2)
print(df[['TeamName', 'W', 'L', 'GP', 'WinPercentage', 'Rank']])
df_sorted = df.sort_values(by='WinPercentage', ascending=False)

print("\nTop Teams by Win Percentage:")
   print(df_sorted[['TeamName', 'WinPercentage', 'Rank']].round(2))
else:
   print("Missing required columns 'Wins', 'Losses', or 'Percentage' in the data.")
```

Response:

___-----

Here is a description of the issues or bugs in the code:

Session Date Error

```
**Line:**
```python
df_team['Season'] = '2022-23'
```

"Mean Wins": mean wins,

\*\*Error:\*\*

It seems like the issue is related to the default season setting in the NBA API. Even though you explicitly set the season in your `TeamInfoCommon` data frame with the `'Season': '2022-23'` line, the data fetched by the API might still be for the current season because the `TeamInfoCommon` class from the `nba\_api` uses the current season as its default parameter. This does not affect the information that you request, but take into account that the last season was '2023-24'.

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
The 'print' statement is missing an 'f' for string interpolation or a proper
concatenation operator. This will raise a **SyntaxError** due to missing `+`
between 'endpoint' and the string. Also, 'e' should be cast to a string (e.g.,
`str(e)`), or it will raise a **TypeError**.
Typo in Class Assignment
 Line:
```python
TeamInfoCommon.nba stats http = CustomNBAStatsHTPP()
 **Error:**
The class name 'CustomNBAStatsHTPP' has a typo with an extra 'P', leading to
potential **AttributeError** or **NameError** due to its undefined nature.
### Misuse of Variable Names
 **Line:**
```python
team_info = TeamInfoCommon(team_name=team_id)
 Error:
The API expects 'team_id', but here 'team_name' is passed as a parameter, which
will raise a **TypeError** or cause an invalid API request.
Incorrect DataFrame Column Handling
 Line:
```python
 if 'Wins' in df.columns and 'Losses' in df.columns and 'PCT' in df.columns:
 **Error:**
The DataFrame likely contains columns named "W" and "L", not "Wins" and
'Losses'. This will raise a **KeyError** because these column names don't exist.
### Missing Column in DataFrame ('TeamName')
 No line associated.
  **Error:**
```

Published using Google Docs

Report abuse

Learn more

New Scratch SFT Coding Examples

pytnon

df['GP'] = df['Wins'] + df['Losses']

Updated automatically every 5 minutes

```
**Error:**
The columns 'Wins' and 'Losses' do not exist; they should be 'W' and 'L'. This
will result in a **KeyError** when trying to access non-existent columns.
### Incorrect Column Name in Statistical Calculations
 **Line:**
```python
mean_wins = round(df['Wins'].mean(), 2)
 Error:
The column 'Wins' is incorrect. It should be 'W', leading to a **KeyError** in all
subsequent statistical operations.
Unnecessary Decimal Precision in Printing
 Line:
```python
....print(f"{stat}: {value:.2f}")
 **Error:**
The print statement includes extra rounding for decimal precision (`:.2f`). This
results in unnecessary code because the values are rounded before this
instruction.
### Incorrect Error Message
 **Line:**
```python
print("Missing required columns 'Wins', 'Losses', or 'Percentage' in the data.")
 Error:
This message is misleading. The correct column names are 'W', 'L', and 'PCT'.
This error message could be confusing.
Here is a functional code that prevents the mentioned issues:
```Python
```

 $20 ext{ of } 45$

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
from requests.packages.urllib3.util.retry import Retry
from requests.exceptions import ReadTimeout, RequestException,
ConnectionError
from urllib3.exceptions import ProtocolError
from http.client import RemoteDisconnected
session = requests.Session()
retry_strategy = Retry(
  total=10.
  backoff factor=0.3,
  status forcelist=[429, 500, 502, 503, 504],
  allowed methods=["HEAD", "GET", "OPTIONS"]
adapter = HTTPAdapter(max_retries=retry_strategy)
session.mount("https://", adapter)
from nba api.stats.library.http import NBAStatsHTTP
class CustomNBAStatsHTTP(NBAStatsHTTP):
  def send api request(self, endpoint, params):
    try:
       return session.get(endpoint, params=params, verify=True, timeout=90)
    except (ReadTimeout, RequestException, RemoteDisconnected,
ProtocolError, ConnectionError) as e:
       print(f"Request failed for URL: {endpoint} with error: {e}")
       return None
TeamInfoCommon.nba stats http = CustomNBAStatsHTTP()
nba teams = teams.get teams()
team ids dict = {team['full name']: team['id'] for team in nba teams}
df = pd.DataFrame()
def fetch team info(team name, team id, retries=3):
  for attempt in range(retries):
       team info = TeamInfoCommon(team id=team id)
       if team info:
         return team info.get data frames()[0]
    except (ReadTimeout, RequestException, RemoteDisconnected,
```

Published using Google Docs

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
at_team = tetcn_team_into(team_name, team_ia)
  if df team is not None:
    df team['TeamName'] = team_name
    df team['Season'] = '2023-24'
    df = pd.concat([df, df team], ignore index=True)
  else:
    print(f"Skipping team {team name} due to repeated failures.")
if 'W' in df.columns and 'L' in df.columns and 'PCT' in df.columns:
  df['GP'] = df['W'] + df['L']
  df['WinPercentage'] = df['PCT'] * 100
  df['Rank'] = df['WinPercentage'].rank(ascending=False)
  mean wins = round(df['W'].mean(), 2)
  median_wins = round(df['W'].median(), 2)
  variance_wins = round(df['W'].var(), 2)
  std_dev_wins = round(df['W'].std(), 2)
  kurtosis wins = round(kurtosis(df['W']), 2)
  team stats = \{
    "Mean Wins": mean_wins,
    "Median Wins": median wins,
    "Variance Wins": variance wins,
    "Standard Deviation Wins": std dev wins,
    "Kurtosis Wins": kurtosis_wins
  print("Overall Team Statistics for Wins (rounded to 2 decimals):")
  for stat, value in team stats.items():
    print(f"{stat}: {value:.2f}")
  print("\nTeam Statistics Summary:")
  df['WinPercentage'] = df['WinPercentage'].round(2)
  print(df[['TeamName', 'W', 'L', 'GP', 'WinPercentage', 'Rank']])
  df sorted = df.sort values(by='WinPercentage', ascending=False)
  print("\nTop Teams by Win Percentage:")
  print(df sorted[['TeamName', 'WinPercentage', 'Rank']].round(2))
else:
  print("Missing required columns 'W', 'L', or 'PCT' in the data.")
```

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

Prompt: I'm studying computer science. Right now, we're studying algorithms for the traveling salesman problem. However, I'm behind in class due to a trip. I need to understand how the method based on a greedy heuristic strategy works. My classmate gave me this code that he developed using different methods to solve the problem for the practical class they had while I was absent:

```
```Python
from sys import maxsize
from itertools import permutations
NUM_VERTICES = 8
def find shortest path cost1(matrix, start point):
 dp = [[maxsize] * NUM_VERTICES for _ in range(1 << NUM_VERTICES)]</pre>
 dp[1 << start point][start point] = 0
 for mask in range(1 << NUM_VERTICES):
 for current city in range(NUM VERTICES):
 if not (mask & (1 << current city)):
 for next_city in range(NUM_VERTICES):
 if mask & (1 << next_city):
 continue
 next mask = mask | (1 << next city)
 dp[next mask][next city] = min(
 dp[next_mask][next_city],
 dp[mask][current_city] + matrix[current_city][next_city]
 final cost = maxsize
 for last city in range(NUM VERTICES):
 if last_city != start_point:
 final cost = min(final cost, dp[(1 << NUM VERTICES) - 1][last city] +
matrix[last city][start point])
 return final cost
def find_shortest_path_cost2(matrix, start_point):
 visited = [False] * NUM VERTICES
```

Report abuse

Learn more

#### New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
tor city in range(NUM_VERTICES):
 if not visited[city] and matrix[current_city][city] < min_distance:
 min_distance = matrix[current_city][city]
 next_city = city
 total cost += min distance
 visited[next city] = True
 current_city = next_city
 total cost += matrix[current city][start point]
 return total cost
def find_shortest_path_cost3(matrix, start_point):
 nodes = []
 for i in range(NUM VERTICES):
 if i != start point:
 nodes.append(i)
 minimum_cost = maxsize
 all permutations = permutations(nodes)
 for perm in all_permutations:
 path cost = 0
 current_node = start_point
 for next node in perm:
 path cost += matrix[current node][next node]
 current node = next node
 path cost += matrix[current node][start point]
 minimum cost = min(minimum_cost, path_cost)
 return minimum_cost
graph matrix = [
 [0, 34, 25, 50, 30, 60, 80, 75],
 [34, 0, 15, 35, 55, 45, 65, 90],
 [25, 15, 0, 30, 40, 20, 55, 85],
 [50, 35, 30, 0, 70, 25, 45, 60],
 [30, 55, 40, 70, 0, 35, 75, 95],
 [60, 45, 20, 25, 35, 0, 50, 70],
 [80, 65, 55, 45, 75, 50, 0, 100],
 [75, 90, 85, 60, 95, 70, 100, 0]
```

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

print("Third method result:", find\_shortest\_path\_cost3(graph\_matrix, start))

Could you tell me how the part of this code that corresponds to this type of strategy solves the problem and to what extent it is efficient?

\_\_\_\_\_

#### Response:

The \*\*Greedy Heuristic Strategy\*\* for solving the Traveling Salesman Problem (TSP) is implemented in the function `find\_shortest\_path\_cost2`. Here is an explanation of how this part of the code works to solve the problem:

- \*\*Initialization\*\*:

A list `visited` is initialized to track whether each city has been visited, with the starting city, given by `start\_point`, marked as visited, while `current\_city` is set to the starting city and `total\_cost` is initialized to 0.

```
"``python
visited = [False] * NUM_VERTICES
visited[start_point] = True
current_city = start_point
total_cost = 0
"``
```

- \*\*Main Loop (Visit All Cities)\*\*:

Here the loop runs `NUM\_VERTICES - 1` times, meaning it visits all cities except the starting city (since the starting city is already visited).

```
```python
for _ in range(NUM_VERTICES - 1):
```

- **Find the Nearest Unvisited City**:

Now, for each iteration of the loop, the algorithm looks for the nearest unvisited city. It initializes `min_distance` to a very large value. The algorithm checks every city, and if the city hasn't been visited and the distance from `current_city` to that city is smaller than `min_distance`, it updates `min_distance` and sets `next_city` to

Published using Google Docs

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
min_aistance = matrix[current_city][city]
next_city = city
```

- **Move to the Nearest City**:

After finding the nearest unvisited city, the distance to the next city is added to `total_cost`. The selected `next_city` is marked as visited in the `visited` array. Finally, `current city` is updated to `next city` for the next iteration.

```
```python
total_cost += min_distance
visited[next_city] = True
current_city = next_city
```

- \*\*Return to the Start Point\*\*:

After visiting all the cities, the algorithm completes the tour by returning to the start point. The distance from the final city back to the starting city is added to 'total cost', and the total path cost is returned.

```
```python
total_cost += matrix[current_city][start_point]
return total_cost
```

With these simple steps, the strategy solves the TSP.

Correctness:

The function `find_shortest_path_cost2` follows the greedy heuristic for the TSP. It ensures each city is visited once and returns to the start but may not find the best solution. It selects the nearest unvisited city at each step, guaranteeing all cities are visited. However, decisions are based on local, not global, information, leading to suboptimal routes, which may not be the shortest path.

Efficiency of the Greedy Heuristic Strategy:

- **Time Complexity**:

The algorithm checks the nearest unvisited city at each step, resulting in a time complexity of O(n²), where `n` is the number of cities. For each of the `n-1` steps, since one city is already visited, the algorithm checks the distances to all unvisited

Published using Google Docs

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

 $\frac{\pi\pi\pi}{\pi}$ Orderly Figure 30 and Cy Emoleticy.

- **Weaknesses**:

The greedy heuristic is not guaranteed to find the optimal solution, as it only makes local, short-term decisions by picking the nearest city, which can lead to suboptimal overall tours. In some cases, the greedy approach may choose a suboptimal city early on, forcing the algorithm to take long detours later.

- **Strengths**:

The greedy algorithm is much faster than the brute-force or dynamic programming approaches, which have exponential time complexities, providing a quick solution for large graphs with many cities. It is also simple to implement, as it doesn't require complex data structures or recursion.

Test Generation Category

Prompt:

As a fitness enthusiast, I've developed an interest in optimizing workout routines based on individual energy levels and available time. To assist with this, I've programmed a Swift function that calculates the recommended intensity level for a workout. This function takes into account the user's current energy level and the total time they have for their workout. I aim to validate the robustness of this function by creating some unit test cases ensuring it can handle various invalid inputs without using external testing libraries. The results should be presented in an easily interpretable manner, perhaps with the use of color-coded text. Invalid inputs include non-numeric energy levels, negative time values, and empty entries.

Here's a sample Swift function that calculates the recommended workout intensity:

func recommendWorkoutIntensity(energyLevel: Int, availableTime: Int) -> String {
 guard energyLevel > 0, availableTime > 0 else {
 return "Invalid input: Energy level and time must be positive numbers."
 }
 let intensityThreshold = (energyLevel + availableTime) / 2

Published using Google Docs

Report abuse

Learn more

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

}

For testing, check for conditions where the energy level or available time are zero or negative, or where non-integer types are mistakenly passed to the function. Please provide me the unit test cases.

Response:

Testing your Swift function `recommendWorkoutIntensity` involves validating its behavior with various inputs, particularly focusing on edge cases and invalid data. Since you're looking to test this without external libraries and desire an easily interpretable, color-coded output, you can write additional Swift code that simulates test cases and uses print statements with ANSI color codes for visual differentiation. Here's a suggestion on how you can implement this:

- 1. **Define a Struct for Test Cases**: Create a structure that holds test case information, including inputs and the expected output.
- 2. **Color-Coded Output**: Use ANSI color codes in the terminal to colorize output based on whether the test passed or failed.
- 3. **Test Runner**: Implement a function to run these tests and check the function's output against expected values.

Here's an example of how you might write this in Swift:

```
```swift
// Define the test case structure
struct TestCase {
 let energyLevel: Any // using 'Any' to allow non-integer inputs
 let availableTime: Any
 let expectedOutput: String
// ANSI color codes for terminal output
let red = "\u{001B}[0;31m"]
let green = "\u{001B}[0;32m"
let reset = "\u\{001B\}[0m']
// Test runner function
func runTests(testCases: [TestCase]) {
 for (index, test) in testCases.enumerated() {
if let energy = test.energyLevel as? Int, let time = test.availableTime as? Int {
let result = recommendWorkoutIntensity(energyLevel: energy, availableTime: time)
 if result == test.expectedOutput {
```

#### **New Scratch SFT Coding Examples**

Updated automatically every 5 minutes

```
integer input provided\(reset)")
// Example test cases
let testCases = [
TestCase(energyLevel: 10, availableTime: 20, expectedOutput: "Moderate Intensity"),
TestCase(energyLevel: -1, availableTime: 30, expectedOutput: "Invalid input: Energy level and time must be positive numbers
TestCase(energyLevel: 30, availableTime: 0, expectedOutput: "Invalid input: Energy level and time must be positive numbers.
 TestCase(energyLevel: "high", availableTime: 45, expectedOutput: "Non-
integer input provided"),
TestCase(energyLevel: 50, availableTime: -5, expectedOutput: "Invalid input: Energy level and time must be positive numbers
TestCase(energyLevel: 0, availableTime: 0, expectedOutput: "Invalid input: Energy level and time must be positive numbers."
runTests(testCases: testCases)
Key Points:
```

- \* \*\*Color Coding\*\*: Red for failed tests and green for passed tests, making it easy to see at a glance which tests are behaving as expected.
- \* \*\*Using `Any` Type\*\*: This allows the simulation of non-integer inputs to check how the function handles them.
- \* \*\*Test Coverage\*\*: The tests include scenarios for negative numbers, zero values, and non-integer inputs, ensuring robust validation. Non-integer input is typically handled through Swift's type checks, but some additional checks were added to the tests to handle that.

This approach will help you confirm the robustness of your function through visual feedback directly in your development environment.

#### **Attention**

New Scratch SFT Coding Examples

Updated automatically every 5 minutes

# ✓**Good Examples**

Category	Prompt	Response	What makes it good:
Test Generation	As a fitness enthusiast, I've developed an interest in optimizing workout routines based on individual energy levels and available time. To assist with this, I've programmed a Swift function that calculates the recommended intensity level for a workout. This function takes into account the user's current energy level and the total time they have for their workout. I aim to validate the robustness of this function by ensuring it can handle various invalid inputs without using external testing libraries. The results should be presented in an easily interpretable manner, perhaps with the use of color-coded text. Invalid inputs include non-numeric energy levels, negative time values, and empty entries.  Here's a sample Swift function that calculates the recommended workout intensity:  '``swift func recommendWorkoutIntensity(energyLevel: Int, availableTime: Int) -> String {     guard energyLevel > 0, availableTime > 0 else {         return "Invalid input: Energy level and time must be positive numbers."	Testing your Swift function 'recommendWorkoutIntensity' involves validating its behavior with various inputs, particularly focusing on edge cases and invalid data. Since you're looking to test this without external libraries and desire an easily interpretable, color-coded output, you can write additional Swift code that simulates test cases and uses print statements with ANSI color codes for visual differentiation. Here's a suggestion on how you can implement this:  1. **Define a Struct for Test Cases**: Create a structure that holds test case information, including inputs and the expected output. 2. **Color-Coded Output**: Use ANSI color codes in the terminal to colorize output based on whether the test passed or failed. 3. **Test Runner**: Implement a function to run these tests and check the function's output against expected values.  Here's an example of how you might write this in Swift:  '``swift  // Define the test case structure struct TestCase { let energyLevel: Any // using 'Any' to allow non-integer inputs	The prompt meets all requirements. R2 is better than R1 since it follows all the prompt requirements.

#### **New Scratch SFT Coding Examples**

Updated automatically every 5 minutes

```
return "Moderate Intensity"
 } else {
 // Test runner function
 return "High Intensity"
 func runTests(testCases: [TestCase]) {
 for (index, test) in testCases.enumerated() {
 if let energy = test.energyLevel as? Int, let
 time = test.availableTime as? Int {
For testing, check for conditions where the
 let result =
energy level or available time are zero or
 recommendWorkoutIntensity(energyLevel:
negative, or where non-integer types are
 energy, availableTime: time)
mistakenly passed to the function.
 if result == test.expectedOutput {
 print("\(green)Test \(index + 1):
 Passed\(reset)")
 } else {
 print("\(red)\ Test\ \(index + 1): Failed -
 Expected \(test.expectedOutput), got
 \(result)\(reset)")
 } else {
 print("\(red)Test \(index + 1): Failed -
 Non-integer input provided\(reset)")
 // Example test cases
 let testCases = [
 TestCase(energyLevel: 10, availableTime:
 20, expectedOutput: "Moderate Intensity"),
 TestCase(energyLevel: -1, availableTime:
 30, expectedOutput: "Invalid input: Energy
 level and time must be positive numbers."),
 TestCase(energyLevel: 30, availableTime: 0,
 expectedOutput: "Invalid input: Energy level
 and time must be positive numbers."),
 TestCase(energyLevel: "high",
 availableTime: 45, expectedOutput: "Non-
 integer input provided"),
 TestCase(energyLevel: 50, availableTime:
 -5, expectedOutput: "Invalid input: Energy
 level and time must be positive numbers."),
 TestCase(energyLevel: 0, availableTime: 0,
```

let reset = "\u{001B}[0m"

2024-10-28, 21:21 31 of 45

<u>(i)</u>

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

		### Key Points:	
		* **Color Coding**: Red for failed tests and green for passed tests, making it easy to see at a glance which tests are behaving as expected.  * **Using `Any` Type**: This allows the simulation of non-integer inputs to check how the function handles them.  * **Test Coverage**: The tests include scenarios for negative numbers, zero values, and non-integer inputs, ensuring robust validation. Non-integer input is typically handled through Swift's type checks, but some additional checks were added to the tests to handle that.  This approach will help you confirm the robustness of your function through visual feedback directly in your development environment.	
Solution reasoning	I have a data analysis task, which is to calculate the correlation between two datasets, dataset A and dataset B. I need to solve the problem in the simplest way possible because I have limited computing and memory resources to work with. My friend suggested this code written in Swift:  import Foundation  func mean(_ data: [Double]) -> Double {     let sum = data.reduce(0, +)     return sum / Double(data.count) }  func covariance(dataA: [Double], dataB: [Double]) -> Double {     let meanA = mean(dataA)     let meanB = mean(dataB)	Your friend has provided you with a method to measure the strength and direction of a linear relationship between two datasets, dataset A and dataset B. This method indicates how changes in one dataset are associated with changes in the other.  Here's an overview of how the provided code calculates the correlation and its efficiency:  * **Simplicity**: The Pearson correlation coefficient is straightforward to compute and widely used in statistics for measuring linear relationships between two datasets.  * **Efficiency**: The time complexity of computing the correlation coefficient is O(n), where n is the number of data points in the datasets. This is because the code involves iterating through the data points to calculate	Both responses are good and follow all the prompt requirements, but the R2 is better because of its presentation.

#### **New Scratch SFT Coding Examples**

Updated automatically every 5 minutes

```
* **Mean Calculation**: The `mean` function
func variance(data: [Double]) -> Double {
 computes the average of the data points in a
 let meanValue = mean(data)
 dataset. This is a simple operation with a time
 var sum: Double = 0
 complexity of O(n) and a space complexity of
 for value in data {
 0(1)
 sum += (value - meanValue) * (value -
meanValue)
 swift
 func mean(data: [Double]) -> Double {
 return sum / Double(data.count - 1)
 let sum = data.reduce(0, +)
 return sum / Double(data.count)
func correlation(dataA: [Double], dataB:
[Double]) -> Double {
 let cov = covariance(dataA: dataA,
 * **Covariance Calculation**: The
dataB: dataB)
 covariance` function measures how much
 let varA = variance(dataA)
 two datasets vary together. It calculates the
 let varB = variance(dataB)
 average product of deviations of each data
 return cov / (sqrt(varA) * sqrt(varB))
 point from their respective means.
                                             ```swift
                                            func covariance(dataA: [Double], dataB:
Please explain how this code calculates
                                            [Double]) -> Double {
the correlation between two datasets and
whether it's efficient.
                                              let meanA = mean(dataA)
                                              let meanB = mean(dataB)
                                              var sum: Double = 0
                                              for i in 0 ..< dataA.count {
                                                sum += (dataA[i] - meanA) * (dataB[i] -
                                            meanB)
                                              return sum / Double(dataA.count - 1)
                                             * **Variance Calculation**: The `variance`
                                            function measures how much each data point
                                            in a dataset deviates from the mean. It is used
                                            to normalize the covariance value. & #x20;
                                            ```swift
 func variance(_ data: [Double]) -> Double {
 let meanValue = mean(data)
```

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples Updated automatically every 5 minutes

* ***Correlation Calculation**: The 'correlation' function combines the covariance and variance functions to compute the Pearson correlation coefficient. It measures the strength and direction of the linear relationship between the datasets:  ''`swift func correlation(dataA: [Double], dataB: [Double]) -> Double { let cov = covariance(dataA; dataA, dataB: dataB) let varA = variance(dataA) let varB = variance(dataB) return cov / (sqrt(varA) * sqrt(varB)) }  ***Correctness:**  This code is both correct and efficient for calculating the correlation coefficient.  The code assumes that both datasets are of equal length; if the datasets are of different lengths, we would need to handle this.  It computes the correlation coefficient with linear time complexity relative to the size of the datasets, providing a reliable measure of the datasets, providing a reliable measure of the linear relationship between datasets A and B.  I am teaching the first steps in simulations. Now I want you to use the example with enumerations to simulate a compass on this webpace hittos://  "Swift  The prompt is detailed and the response			***	
func correlation(dataA: [Double], dataB: [Double]) -> Double {     let cov = covariance(dataA: dataA, dataB: dataB)     let varA = variance(dataA)     let varB = variance(dataB)     return cov / (sqrt(varA) * sqrt(varB)) }  **Correctness:**  This code is both correct and efficient for calculating the correlation coefficient.   The code assumes that both datasets are of equal length; if the datasets are of different lengths, we would need to handle this.  It computes the correlation coefficient with linear time complexity relative to the size of the datasets, providing a reliable measure of the linear relationship between datasets A and B.  I am teaching the first steps in simulations. Now I want you to use the example with enumerations to simulate  The prompt is detailed and specific, and specific, and			`correlation` function combines the covariance and variance functions to compute the Pearson correlation coefficient. It measures the strength and direction of the	
This code is both correct and efficient for calculating the correlation coefficient.   The code assumes that both datasets are of equal length; if the datasets are of different lengths, we would need to handle this.  It computes the correlation coefficient with linear time complexity relative to the size of the datasets, providing a reliable measure of the linear relationship between datasets A and B.  I am teaching the first steps in simulations. Now I want you to use the example with enumerations to simulate  This code is both correct and efficient for calculating the correlation coefficient.   The code assumes that both datasets are of equal length; if the datasets are of different lengths, we would need to handle this.  It computes the correlation coefficient with linear time complexity relative to the size of the datasets, providing a reliable measure of the linear relationship between datasets A and B.			<pre>func correlation(dataA: [Double], dataB: [Double]) -&gt; Double {     let cov = covariance(dataA: dataA, dataB:     dataB)     let varA = variance(dataA)     let varB = variance(dataB)     return cov / (sqrt(varA) * sqrt(varB)) }</pre>	
calculating the correlation coefficient.   The code assumes that both datasets are of equal length; if the datasets are of different lengths, we would need to handle this.  It computes the correlation coefficient with linear time complexity relative to the size of the datasets, providing a reliable measure of the linear relationship between datasets A and B.  I am teaching the first steps in simulations. Now I want you to use the example with enumerations to simulate  The prompt is detailed and specific, and				
equal length; if the datasets are of different lengths, we would need to handle this.  It computes the correlation coefficient with linear time complexity relative to the size of the datasets, providing a reliable measure of the linear relationship between datasets A and B.  I am teaching the first steps in simulations. Now I want you to use the example with enumerations to simulate  Here is the app that simulates a compass with the required specifications:  The prompt is detailed and specific, and				
linear time complexity relative to the size of the datasets, providing a reliable measure of the linear relationship between datasets A and B.  I am teaching the first steps in simulations. Now I want you to use the example with enumerations to simulate  linear time complexity relative to the size of the datasets, providing a reliable measure of the datasets, providing a reliable measure of the datasets, providing a reliable measure of the datasets A and B.  The prompt is detailed and specific, and			equal length; if the datasets are of different	
simulations. Now I want you to use the example with enumerations to simulate with the required specifications: detailed and specific, and			linear time complexity relative to the size of the datasets, providing a reliable measure of the linear relationship between datasets A and	
docs.swift.org/swift-book/ import Foundation addresses	RAG	simulations. Now I want you to use the example with enumerations to simulate a compass on this webpage <a href="https://">https://</a>	with the required specifications: ""Swift	detailed and specific, and the response

#### **New Scratch SFT Coding Examples**

Updated automatically every 5 minutes

Also, when the user leaves the case west = "West" application, the last location is given. Just like the example on the web // Methods to change the direction mutating func turnNorth() { page, show the total number of self = .north selections, as well as the number of times each direction was chosen and an associated probability, with two mutating func turnSouth() { decimal places. self = .south mutating func turnEast() { self = .eastmutating func turnWest() { self = .west// Method to change the direction based on input mutating func turn(direction: String) { switch direction.lowercased() { case "north": self.turnNorth() case "south": self.turnSouth() case "east": self.turnEast() case "west": self.turnWest() default: print("Invalid direction. Please enter North, South, East, or West.") // Initialize the current direction and remember the initial direction var currentDirection = CompassPoint.west let rememberedDirection = currentDirection

Published using Google Docs

Report abuse

Learn more

#### New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
print("\nCurrent direction is
\(currentDirection.rawValue).")
 print("Enter a direction to turn (North,
South, East, West), type 'random' for a
random direction, or type 'exit' to quit:")
// Function to read user input
func readInput() -> String? {
 return readLine()
// Function to get a random direction
func getRandomDirection() ->
CompassPoint {
 return
CompassPoint.allCases.randomElement()!
// Main loop to handle user input
while true {
 promptUser()
 if let input = readInput() {
 if input.lowercased() == "exit" {
 break // Exit the loop if user types
'exit'
 } else if input.lowercased() ==
"random" {
 currentDirection =
getRandomDirection()
 directionCount[currentDirection,
default: 0] += 1
 print("Random direction selected:
\(currentDirection.rawValue)")
 } else {
 currentDirection.turn(direction:
input)
 if let direction =
CompassPoint(rawValue:
currentDirection.rawValue) {
 directionCount[direction, default:
0] += 1
```

ιαπο ρισπηρισσει () ξ

37 of 45

Published using Google Docs

Report abuse

Learn more

#### **New Scratch SFT Coding Examples** Updated automatically every 5 minutes directionCount.values.reduce(U, +) print("\nTotal number of selections: \(totalSelections)") // Print direction selection details print("\nDirection Selection Details:") for (direction, count) in directionCount { let probability = totalSelections > 0 ? (Double(count) / Double(totalSelections) \* 100): 0.0 print("\(direction.rawValue): \(count) selections, Probability: \(String(format: "%.2f", probability))%") // Print the final and remembered directions print("\nFinal direction is \(currentDirection.rawValue).") print("Remembered direction was \(rememberedDirection.rawValue).") The program defines the 'CompassPoint' enum with direction methods ('turnNorth', `turnSouth`, `turnEast`, `turnWest`, `turn`), based on the web page provided. It initializes current and remembered directions, tracks direction counts, prompts the user for input with the 'promptUser' function, reads user input with the `readInput` function, and gets a random direction with the 'getRandomDirection' function. Processes user input in a loop

I am working on a project that seeks to

implement a software solution for the

calculates and prints the total selections and probabilities, and displays the final

According to the information gathered from

medical/elf/courses/view/146858/ecg-

interpretation/1/page15>", there are three

the shared source, "<https://www.abdn.ac.uk/

and remembered directions.

identification and classification of cardiovascular diseases. I am trying to

RAG

2024-10-28, 21:21

The code

correctly

retrieves the information

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

	fft that stores the types lescriptions in a ts them.  // ty va "E ec pa ca ca for the type of type of type of the type of type of the type of type	f ECGs and their characteristics:  '``swift 'A dictionary is created to store the ECG ypes and their descriptions ar ECG_types: [String: String] = ["Regular": Equally spaced", "Regularly irregular": "Not qually spaced but there is a recognized attern", "Irregularly irregular": "No pattern an be identified"]  'I tterate through the dictionary to print the alues or (ECG_type, description) in ECG_types {     print("\(ECG_type): \(description)")   the webpage summarizes three different hythms that can provide different insights into ECG interpretations.	
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

# **○**Bad Prompt Examples

Category	Prompt	Response	What makes it bad:
Bug Fixing Prompt	I have a code snippet in Swift that has an error. The code snippet receives a JSON object, extracts the values, and displays it in a sentence string output. There is a decoding error when I run the code. Can you please debug and provide a solution for the code? Here is the code snippet:	Certainly! The code snippet has an error within this line:  let person = try JSONDecoder().decode(Person.self, from: jsonData)  The issue is that the Person struct is expecting age to be an Int, but jsonPersonString has age as a string.	The actual error message is not shared in the prompt. It is not complex enough.

age . ou,

#### New Scratch SFT Coding Examples

Updated automatically every 5 minutes

```
"location": "New York"
struct Person: Codable {
 let firstName: String
 let lastName: String
 let age: Int
 To further improve the code, we can
 add comments and some error
 let location: String
 handling with messages to help
 with debugging and to catch future
func displayPersonInfo(from
 errors. Here is the revised code
jsonString: String) {
 snippet:
 let isonData =
jsonString.data(using: .utf8)!
 import Foundation
 do {
 // JSON string with person's info
 let jsonPersonString = """
let person = try
JSONDecoder().decode(Person.self,
 "firstName": "John",
from: jsonData)
 "lastName": "Doe",
 let output =
"\(person.firstName)
 "age": 30.
\(person.lastName) is \(person.age)
 "location": "New York"
years old and lives in
(person.location)."
 print(output)
 // Person Model
 } catch {
 print("Failed to decode JSON:
 struct Person: Codable {
 let firstName: String
\(error)")
 let lastName: String
 let age: Int
 let location: String
displayPersonInfo(from:
isonPersonString)
 // Function to decode JSON and
 display the person's information
 // Input: JSON string
 // Output: String containing person's
 func displayPersonInfo(from
 jsonString: String) {
 // Convert JSON string to Data
 guard let jsonData =
 jsonString.data(using: .utf8) else {
print("Failed to convert string to
 data.")
 return
```

# New Scratch SFT Coding Examples

Updated automatically every 5 minutes

		\(\(\(\person\).\(\alpha\) and lives in \(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(\person\).\(	
Solution Reasoning Prompt	I am an experienced Swift developer, and I would like to learn the reasoning behind the correct and more efficient approaches to the solution. I have implemented a function in Swift that takes a list of integers and then returns a transformed list based on certain rules. The problem I am facing right now is, I don't think this implementation is the most efficient or correct way to solve the problem. This function tries to double even numbers and triple odd numbers.  The Swift code corresponding to the above is given below:  func transformNumbers(_numbers: [Int]) -> [Int] {	Analysis and Review of the Solution:  The function transformNumbers(_:) written in Swift is expected to transform a list of integers by the rules: even numbers have to become doubled, and odd numbers have to become tripled. Below, I break down and evaluate the code discussing its correctness and efficiency, and I propose possible improvements. func transformNumbers(_ numbers: [Int]) -> [Int] {    var result = [Int]() // Initialize an empty array to store the results for number in numbers {    if number % 2 == 0 {       // Even number: multiply by 2 result.append(number * 2)	The code in the task is extremely simple. We need to be more creative and create a prompt with some challenges to the LLM.

#### New Scratch SFT Coding Examples

Updated automatically every 5 minutes

return result }

It is an analysis of the problem, rather closely done. Autolycus describes firstly how the code changes a list of integers, that's what it means for code to multiply even number by 2 and odd by 3. Second, indicate that the solution is indeed applying the transformation rules described, then explain the kinds of issues or edge cases that ought to be tested for; for example, handling huge numbers that could cause overflows and allowing arrays that can be empty with nil values. Also, describe the time and space complexity of the solution: efficient or, again, how something could be done in a better way to arrive at the same result. Finally, optimization suggestions and/or alternative approaches that can be offered to make a solution more performant or clear: breaking down the solution into meaningful functions or methods to increase the readability and maintainability of the code. This should create a deep understanding of the problem and solution, allowing one to write code that is correct, efficient, and clear.

#### Solution Analysis:

The program iterates through an integer array, checks the parity of each integer, and, according to the base number being even or odd, multiplies accordingly, i.e., if the number is even, it multiplies by 2. Now whenever it will find odd numbers, it will multiply odd numbers by 3 and insert the result into the array of outcome—but only whenever the number is odd.

Correctness Evaluation: In essence, the function being tested has the appropriate specifications for transformation, but it has failed to take proper care of errors and boundary conditions.

#### For instance:

Overflow Handling: Function should handle possible overflow in case of products of very large numbers.

Empty Input: The function should properly handle an empty input array.

Published using Google Docs

Report abuse

Learn more

# **New Scratch SFT Coding Examples**

Updated automatically every 5 minutes

Efficiency Consideration: The time complexity of the solution is in O(n), where n refers to the number of elements in the input array, because all elements would be covered once. Space complexity is also in O(n) and that is because of the result array in which we will store the transformed values. By and large, however, the function seems pretty effective for this purpose although we may take a more functional approach to enhance readability and gain possible performance benefits.

Better Solution:

Below is the improved version of the function with error handling, optimizations, and in-depth comments:

func transformNumbers(\_ numbers:
[Int]?) -> [Int]? {

// Check for nil or empty input

guard let numbers = numbers,!
numbers.isEmpty else {

print("Input array is nil or empty.")

return nil

Published using Google Docs

Report abuse

Learn more

# New Scratch SFT Coding Examples

#### Updated automatically every 5 minutes

	//
	approach and to enhance readability
	let result: [Int] = numbers.map { number in
	if number % 2 == 0 {
	// Check for potential overflow when multiplying
	guard number <= Int.max / 2 else {
	print("Overflow detected for even number: \(number)")
	return 0
	}
	return number * 2
	} else {
	// Check for potential overflow when multiplying
	guard number <= Int.max / 3 else {

43 of 45

Published using Google Docs

Report abuse

Learn more

#### **New Scratch SFT Coding Examples**

Updated automatically every 5 minutes

return number \* 3 return result Improvements to the Key: Error Handling: The function now can handle nil and empty inputs, take graceful exits with error messages; it will also help handle probable overflows in multiplication by checking if the multiplication will go beyond the maximum allowed integer value. integér value. Functional Approach: The use of map makes it more functional and provides a very concise way of applying the transform to all elements, thereby making it more readable. Detailed Comments: There are comments in the code explaining each step, which helps understand the logic and intention.

44 of 45

i Published using Google Docs		Report	t abuse	Learn more
New Scratch SFT Coding Examples		Updated	d automatically	v every 5 minutes