COSC2429_Assignment1_2021C

1. Find the 80% Split Number

Objective: To find a number in a list of 20 integers where at least 80% of the numbers are equal to or smaller than it.

- **Input**
  - List of 20 integers.
- **Output**
  - A single integer from the list.

Steps:

1. **Understand the List**: A list in programming is a collection of items (in this case, integers) that are ordered and changeable.
2. **Sort the List**: To find the 80% split number, you first need to sort the list in ascending order.
3. **Find the Index**: Calculate the index corresponding to 80% of the length of the list. In a list of 20 items, this would be the 16th item (since 80% of 20 is 16).
4. **Retrieve the Number**: Fetch the number at this index from the sorted list.
5. **Return the Number**: This number is the one where 80% of the numbers in the list are equal to or less than it.

```
def find_split_80(integer_list):
    if not integer_list or len(integer_list) != 20:
        raise ValueError("The list must contain exactly 20 integers.")

    sorted_list = sorted(integer_list)
    # Finding the number where 80% of numbers are equal to or smaller than it
    index = int(len(sorted_list) * 0.8) - 1
    return sorted_list[index]
```

2. Estimate π with Random Points

Objective: To estimate the value of π by generating random points and checking how many fall inside a unit circle.the value of π.

- **Input**
  - Number of random points to generate.
- **Output**
  - Estimated value of π.

Steps:

1. **Understand the Concept**: Imagine a square with a circle inside it. The ratio of the area of the circle to the square can be used to estimate π.
2. **Generate Points**: Use a random number generator to create points within the square.
3. **Check Points Inside Circle**: For each point, check if it lies inside the circle (distance from center ≤ radius, which is 1).
4. **Calculate π**: The ratio of points inside the circle to the total points, multiplied by 4, gives an estimate of π.
5. **Return the Estimate**: The calculated value is your estimated π.

```
import random
import math

def estimate_pi(num_points):
    """
    Estimates the value of pi using the Monte Carlo method.

    Parameters:
    num_points (int): The number of random points to generate.

    Returns:
    float: The estimated value of pi.
    """

    # Initialize the count of points inside the circle
    points_inside_circle = 0

    # Generate points and count how many fall inside the unit circle
    for _ in range(num_points):
        x, y = random.uniform(-1, 1), random.uniform(-1, 1)  # Generate random x, y coordinates
        distance = math.sqrt(x**2 + y**2)  # Calculate the distance from the origin
        if distance <= 1:
            points_inside_circle += 1  # Point is inside the circle

    # Calculate the estimated pi
    estimated_pi = 4 * points_inside_circle / num_points

    return estimated_pi

# Example usage:
# Estimate pi using 1,000,000 random points
estimated_pi = estimate_pi(1000000)
estimated_pi
```

```
3.142856
```

3. Calculate Flour Order for Pizzas

Objective: To calculate the total amount of flour needed based on the number of pizzas and choose the best flour supplier.

- **Input**
  - Number of large thick, large thin, medium thick, and medium thin pizzas.
- **Output**
  - Total flour required (rounded to nearest 2kg),
  - Chosen flour provider (A or B),
  - Total cost.

Steps:

1. **Understand Pizza Types**: Know how much flour each type of pizza needs.
2. **Calculate Total Flour**: Multiply the number of each type of pizza by the flour it requires, then sum these amounts.
3. **Add Wastage**: Add 6% to the total for wastage.
4. **Round Off**: Flour is ordered in 2kg increments, so round off to the nearest 2kg.
5. **Compare Suppliers**: Calculate the cost from two suppliers and choose the cheaper one.
6. **Return the Order Details**: Include total flour needed, chosen supplier, and total cost.

```python
def flour_order(large_thick, large_thin, medium_thick, medium_thin):
    """
    This function calculates the amount of flour needed for a given number of pizzas
    of different sizes and thicknesses, and determines the cost and provider to purchase from.

    Parameters:
    large_thick (int): The number of large thick pizzas.
    large_thin (int): The number of large thin pizzas.
    medium_thick (int): The number of medium thick pizzas.
    medium_thin (int): The number of medium thin pizzas.

    Returns:
    tuple: A tuple containing the amount of flour to order, the provider selected, and the total cost.
    """

    # Define the amount of flour needed for each type of pizza (in grams)
    flour_per_large_thick = 550
    flour_per_large_thin = 500
    flour_per_medium_thick = 450
    flour_per_medium_thin = 400

    # Calculate the total amount of flour needed for the order (in grams)
    total_flour = (
        large_thick * flour_per_large_thick +
        large_thin * flour_per_large_thin +
        medium_thick * flour_per_medium_thick +
        medium_thin * flour_per_medium_thin
    )

    # Account for the 6% waste of flour
    total_flour *= 1.06

    # Convert the total flour needed to kilograms and round up to the nearest 2kg
    total_flour_kg = math.ceil(total_flour / 2000) * 2

    # Calculate the cost from provider A and B before discount
    cost_a = total_flour_kg * 30000
    cost_b = total_flour_kg * 31000

    # Apply discounts based on the amount of flour ordered
    if total_flour_kg < 30:
        cost_a *= 0.97  # 3% discount for A
    else:
        cost_a *= 0.95  # 5% discount for A

    if total_flour_kg < 40:
        cost_b *= 0.95  # 5% discount for B
    else:
        cost_b *= 0.90  # 10% discount for B

    # Determine the provider to buy from based on cost
    if cost_a <= cost_b:
        selected_provider = 'A'
        total_cost = cost_a
    else:
        selected_provider = 'B'
        total_cost = cost_b

    # Print the statement required by the problem
    print(f"We need to order {total_flour_kg}kg of flour, which costs {cost_a}VND if we buy from A and {cost_b}VND if we buy from B.")

    return total_flour_kg, selected_provider, total_cost

# Example usage of the function
flour_order_result = flour_order(10, 20, 30, 40)
```

```
We need to order 48kg of flour, which costs 1368000.0VND if we buy from A and 1339200.0VND if we buy from B.
```

The function `flour_order` has been executed with an example order of pizzas, and it has determined the following:

- The restaurant needs to order 48kg of flour.
- If purchased from Provider A, the cost would be 1,368,000 VND.
- If purchased from Provider B, the cost would be 1,339,200 VND.

Thus, the function has selected Provider B as the cheaper option and returned the amount of flour to order, the selected provider, and the total cost. Here are the results in detail:

- Order amount: 48kg
- Selected provider: B

- Total cost: 1,339,200 VND

This decision is based on the total cost being lower from Provider B, despite Provider A being more friendly. If there had been a tie in cost, the function is designed to select Provider A.

4. Draw a Stacked Bar Chart with Turtle

Objective: To create a visual stacked bar chart representing the number of different types of pizzas made on a specific date.

- **Input**
  - Record date,
  - Number of large thick, large thin, medium thick, and medium thin pizzas.
- **Output**
  - A stacked bar chart drawn using Turtle graphics.

Steps:

1. **Setup Turtle**: Learn the basics of the Turtle module for drawing.
2. **Draw Bars**: For each pizza type, draw a colored rectangle representing its count.
3. **Stack Bars**: Each new bar starts where the previous one ended.
4. **Label the Chart**: Add the record date and total pizzas at the appropriate positions.
5. **Complete the Drawing**: Finalize the drawing and ensure it visually represents the data correctly.

Ref: Graphical Python Programming For Beginners with Turtle

```python
# Python program to draw a turtle
import turtle as t


def draw_bar_chart(record_date, large_thick, large_thin, medium_thick, medium_thin):
    width = 50
    gap_size = 20
    arrow_size = 8
    total = large_thick + large_thin + medium_thick + medium_thin
    t.getscreen().bgcolor("white")

    def draw_bar(height, color="black"):
        t.color(color)
        t.begin_fill()
        t.forward(width)
        t.left(90)
        t.forward(height)
        t.left(90)
        t.forward(width)
        t.end_fill()

    def draw_triangle():
        t.begin_fill()
        t.left(90)
        t.forward(arrow_size / 2)
        t.right(120)
        t.forward(arrow_size)
        t.right(120)
        t.forward(arrow_size)
        t.right(120)
        t.end_fill()

    # Draw the x and y axes
    t.color("black")
    t.forward(4 * gap_size + width)
    draw_triangle()
    t.penup()
    t.goto(0, 0)
    t.pendown()
    t.forward(total+gap_size)
    draw_triangle()

    t.penup()
    t.goto(gap_size, 0)
    t.left(180)
    t.pendown()

    # Draw Bars
    draw_bar(large_thick, "#d81c18")
    t.left(180)
    draw_bar(large_thin, "#e9963a")
    t.left(180)
    draw_bar(medium_thick, "#fdc70c")
    t.left(180)
    draw_bar(medium_thin, "#fff33b")

    # The total number of pizzas must be printed on top of the bar
    t.penup()
    t.right(90)
    t.forward(10)
    t.pendown()
    t.color("black")
    t.write(str(total), font=('', 12, ''))

    # The record date must be printed below the bar.
    t.penup()
    t.left(180)
    t.forward(total + 2 * gap_size)
    t.left(90)
    t.forward(width / 2)
    t.write(record_date, align="center", font=('', 12, ''))
    t.pendown()
    t.hideturtle()
    t.done()

# Example usage of the function
draw_bar_chart("26/12/2021", 30, 40, 20, 30)

```