# PDSA Week - 1 Live Coding

## 📃 Live Coding Problem 1

A positive integer `m` is a prime product if it can be written as `p × q`, where `p` and `q` are both primes. .

Write a Python function **primeproduct(m)** that takes an integer `m` as input and returns `True` if `m` is a prime product and `False` otherwise.

(If `m` is not positive, function should return `False`.)

**Sample Input**

```
6
```

**Output**

```
True
```

## 👉 Problem Explanation

We need to determine if a given positive integer `m` is a prime product. A prime product means that `m` can be written as the product of two prime numbers, `p` and `q`.

For example:

- For `m = 6`, we can write it as `2 * 3`, and both 2 and 3 are prime numbers. So, the function should return `True`.
- For `m = 8`, we cannot write it as a product of two prime numbers. So, the function should return `False`.

## 👉 Solution Approach

1. **Define a function `is_prime(n)` to check if `n` is a prime number:**
    - If `n` is less than 2, return `False`.
    - Check divisibility from 2 to the square root of `n`. If any number divides `n`, return `False`.
    - If no divisors are found, return `True`.
2. **Define the main function `primeproduct(m)`:**
    - Returns `False` immediately if `m` is not a positive integer.

- Iterates through all numbers from 2 to the square root of `m`.
- For each number `i`, checks if `i` is a divisor of `m` and if both `i` and `m // i` are prime.
- If such a pair of primes is found, returns `True`.
- If no such pair is found after the loop, returns `False`.

In [38]:
```python
def is_prime(n):
  """Helper function to check if n is a prime number."""
  if n <= 1:
    return False
  for i in range(2, int(n ** 0.5) + 1):
    if n % i == 0:
      return False
  return True


print(is_prime(2))
print(is_prime(8))
```

```
True
False
```

In [46]:
```python
def primeproduct(m):
  """Function to check if m is a prime product."""
  if m <= 0:
    return False
  for i in range(2, int(m ** 0.5) + 1):
    if m % i == 0:  # i is a divisor of m
      if is_prime(i) and is_prime(m // i):
        return True
  return False


print(primeproduct(6))   # Output: True (2*3)
print(primeproduct(9))   # Output: True (3*3)
print(primeproduct(15))  # Output: True (3*5)

print(primeproduct(1))   # Output: False
print(primeproduct(8))   # Output: False
print(primeproduct(12))  # Output: False
```

```
True
True
True
False
False
False
```

## Live Coding Problem 2

Write a function **del_char(s,c)** that takes strings `s` and `c` as input, where `c` has length 1 (i.e., a single character), and returns the string obtained by deleting all occurrences of `c` in `s`.

If `c` has length other than 1, the function should return `s`.

**Sample input-1**

```
banana
b
```

**Output**

```
anana
```

**Sample input-2**

```
banana
an
```

**Output**

```
banana
```

## 👉 Problem Explanation

We need to remove all occurrences of a given character `c` from a string `s`. The function should handle two cases:

1. If `c` is a single character, remove all occurrences of `c` from `s`.
2. If `c` is not a single character, return the original string `s` without any changes.

For example:

- For `s = "banana"` and `c = "b"`, we should remove all 'b' characters from "banana", resulting in "anana".
- For `s = "banana"` and `c = "an"`, since `c` is not a single character, the function should return the original string "banana".

## 👉 Solution Approach

**Define the function `del_char(s, c)`:**

- If `c` has a length other than 1, return `s`.
- Initialize an empty string `result`.
- Iterate through each character in `s`:
    - If the character is not equal to `c`, add it to `result`.
- Return `result`.

In [47]:
```
def del_char(s, c):
  """Function to delete all occurrences of character c from string s."""
  if len(c) != 1:
    return s
  result = ''
  for char in s:
    if char != c:
      result += char
  return result


print(del_char("banana", "b"))   # Output: "anana"
print(del_char("banana", "a"))   # Output: "bnn"
print(del_char("banana", "n"))   # Output: "baaa"
print(del_char("banana", "an"))  # Output: "banana"

print(del_char("apple", "p"))    # Output: "ale"
print(del_char("apple", ""))     # Output: "apple"
```

```
anana
bnn
baaa
banana
ale
apple
```

# 📃 Live Coding Problem 3

Write a function **shuffle(l1,l2)** that takes two lists, `l1` and `l2` as input, and returns a list consisting of the first element in `l1`, then the first element in `l2`, then the second element in `l1`, then the second element in `l2`, and so on.

If the two lists are not of equal length, the remaining elements of the longer list are appended at the end of the shuffled output.

**Sample Input**

```
    [0,2,4]
    [1,3,5]
```

**Output**

```
[0, 1, 2, 3, 4, 5]
```

**Sample Input**

```
    [0,2,4]
    [1]
```

**Output**

```
[0, 1, 2, 4]
```

## 👉 Problem Explanation

We need to create a new list by alternating elements from two input lists, `l1` and `l2`. If the lists are of unequal length, any remaining elements from the longer list should be added to the end of the new list.

For example:

- For `l1 = [0, 2, 4]` and `l2 = [1, 3, 5]`, the function should return `[0, 1, 2, 3, 4, 5]`.
- For `l1 = [0, 2, 4]` and `l2 = [1]`, the function should return `[0, 1, 2, 4]`.

## 👉 Solution Approach

**Define the function `shuffle(l1, l2)`:**

- Initialize an empty list `result`.
- Use a loop to iterate through the indices of the shorter list:
  - Append the element from `l1` at the current index to `result`.
  - Append the element from `l2` at the current index to `result`.

- Append the remaining elements from the longer list to `result`.
- Return `result`.

In [41]:
```python
def shuffle(l1, l2):
  """Function to shuffle two lists by alternating their elements."""
  result = []
  min_length = min(len(l1), len(l2))

  for i in range(min_length):
    result.append(l1[i])
    result.append(l2[i])

  result.extend(l1[min_length:])
  result.extend(l2[min_length:])

  return result


print(shuffle([0, 2, 4], [1, 3, 5]))  # Output: [0, 1, 2, 3, 4, 5]
print(shuffle([0, 2, 4], [1]))        # Output: [0, 1, 2, 4]
print(shuffle([0], [1, 3, 5]))        # Output: [0, 1, 3, 5]
print(shuffle([], [1, 3, 5]))         # Output: [1, 3, 5]
print(shuffle([0, 2, 4], []))         # Output: [0, 2, 4]
```

```
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 4]
[0, 1, 3, 5]
[1, 3, 5]
[0, 2, 4]
```

## 📃 Live Coding Problem 4

Write a function **expanding(L)** that takes a list of integer `L` as input and returns `True` if the absolute difference between each adjacent pair of elements strictly increases.

**Sample Input**

```
[1,3,7,2,9]
```

**Output**

`True`

**Sample Input**

```
[1,3,7,2,-3]
```

**Output**

`False`

## 👉 Problem Explanation

We need to determine if the absolute differences between each pair of adjacent elements in the list `L` are strictly increasing. This means that for every adjacent pair of elements in `L`, the difference between the current pair should be greater than the difference between the previous pair.

For example:

- For `L = [1, 3, 7, 2, 9]`, the absolute differences between adjacent pairs are `[2, 4, 5, 7]`. These differences are strictly increasing, so the function should return `True`.
- For `L = [1, 3, 7, 2, -3]`, the absolute differences between adjacent pairs are `[2, 4, 5, 5]`. The differences do not strictly increase, so the function should return `False`.

## 👉 Solution Approach

**Define the function `expanding(L)`:**

- If the list `L` has fewer than 2 elements, return `True` (since there's no pair to compare).
- Initialize a list `diffs` to store the absolute differences.
- Iterate through the list `L` to compute the absolute differences and store them in `diffs`.
- Iterate through `diffs` to check if each difference is greater than the previous one.
- If all differences are strictly increasing, return `True`; otherwise, return `False`.

In [42]:
```
def expanding(L):
  """Function to check if the absolute differences between adjacent elements are st
  if len(L) < 2:
    return True

  diffs = []
  for i in range(1, len(L)):
    diffs.append(abs(L[i] - L[i - 1]))

  for i in range(1, len(diffs)):
    if diffs[i] <= diffs[i - 1]:
      return False

  return True


print(expanding([1, 3, 7, 2, 9]))     # Output: True, diffs = [2, 4, 5, 7]
print(expanding([1, 3, 7, 2, -3]))    # Output: False, diffs = [2, 4, 5, 5]

print(expanding([1]))                 # Output: True, diffs = []
print(expanding([]))                  # Output: True, diffs = []

print(expanding([10, 1, 10, 1, 10]))  # Output: False, diffs = [9, 9, 9, 9]
```

```
True
False
True
True
False
```

## 📃 Live Coding Problem 5

Write a Python function `sumsquare(l)` that takes a nonempty list of integers as input and returns a list `[odd,even]`, where `odd` is the sum of squares all the odd numbers in `l` and `even` is the sum of squares of all the even numbers in `l`.

**Sample Input**

```
[1,3,5]
```

**Output**

```
[35, 0]
```

**Sample Input**

```
[-1,-2,3,7]
```

**Output**

```
[59, 4]
```

## 👉 Problem Explanation

We need to calculate two values from a given list of integers:

1. The sum of squares of all the odd numbers.
2. The sum of squares of all the even numbers.

For example:

- For `l = [1, 3, 5]`, the sum of squares of odd numbers is `1^2 + 3^2 + 5^2 = 35` and there are no even numbers, so the function should return `[35, 0]`.
- For `l = [-1, -2, 3, 7]`, the sum of squares of odd numbers is `(-1)^2 + 3^2 + 7^2 = 59` and the sum of squares of even numbers is `(-2)^2 = 4`, so the function should return `[59, 4]`.

## 👉 Solution Approach

**Define the function `sumsquare(l)`:**

- Initialize two variables `odd_sum` and `even_sum` to 0.
- Iterate through each element in `l`:

    - If the element is odd, add its square to `odd_sum`.
    - If the element is even, add its square to `even_sum`.
- Return the list `[odd_sum, even_sum]`.

In [43]:
```python
def sumsquare(l):
  """Function to return the sum of squares of odd and even numbers in the list."""
  odd_sum = 0
  even_sum = 0

  for num in l:
    if num % 2 == 0:
      even_sum += num ** 2
    else:
      odd_sum += num ** 2

  return [odd_sum, even_sum]


print(sumsquare([1, 3, 5]))          # Output: [35, 0]
print(sumsquare([-1, -3, -5]))       # Output: [35, 0]
print(sumsquare([1, 2, 3, 4, 5]))    # Output: [35, 20]

print(sumsquare([-2, -4, -6]))       # Output: [0, 56]
print(sumsquare([0, 2, 4, 6]))       # Output: [0, 56]
print(sumsquare([-1, -2, 3, 7]))     # Output: [59, 4]
```

```
[35, 0]
[35, 0]
[35, 20]
[0, 56]
[0, 56]
[59, 4]
```

## Live Coding Problem 6

Write a Python function `histogram(l)` that takes as input a list of integers with repetitions and returns a list of pairs as follows:.

- for each number `n` that appears in `l`, there should be exactly one pair `(n,r)` in the list returned by the function, where `r` is the number of repetitions of `n` in `l`.
- the final list should be sorted in ascending order by `r`, the number of repetitions. For numbers that occur with the same number of repetitions, arrange the pairs in ascending order of the value of the number.

**Sample Input**

```
[13,12,11,13,14,13,7,7,13,14,12]
```

**Output**

```
[(11, 1), (7, 2), (12, 2), (14, 2), (13, 4)]
```

**Sample Input**

```
[13,7,12,7,11,13,14,13,7,11,13,14,12,14,14,7]
```

**Output**

```
[(11, 2), (12, 2), (7, 4), (13, 4), (14, 4)]
```

## 👉 Problem Explanation

We need to create a histogram of the occurrences of each number in a given list of integers. The histogram should consist of pairs `(n, r)` where `n` is a unique number in the list, and `r` is the number of times `n` appears in the list.

For example:

- For `l = [13, 12, 11, 13, 14, 13, 7, 7, 13, 14, 12]`, the histogram should be `[(11, 1), (7, 2), (12, 2), (14, 2), (13, 4)]`.
- For `l = [13, 7, 12, 7, 11, 13, 14, 13, 7, 11, 13, 14, 12, 14, 14, 7]`, the histogram should be `[(11, 2), (12, 2), (7, 4), (13, 4), (14, 4)]`.

## 👉 Solution Approach

**Define the function `histogram(l)`:**

- Initialize an empty dictionary `count`.
- Iterate through each number `n` in the list `l`:
    - Increment the count of `n` in the dictionary.
- Convert the dictionary items to pairs `(n, r)`.
- Sort the pairs first by `r` in ascending order, and then by `n` in ascending order.
- Return the sorted list of pairs.

In [44]:

```python
def histogram(l):
  """Function to create a histogram of occurrences of each number in the list."""
  count = {}
  for num in l:
    count[num] = count.get(num, 0) + 1

  hist = [(num, count[num]) for num in count]
  hist.sort(key=lambda x: (x[1], x[0]))
  return hist


# Output: [(11, 1), (7, 2), (12, 2), (14, 2), (13, 4)]
print(histogram([13, 12, 11, 13, 14, 13, 7, 7, 13, 14, 12]))

# Output: [(11, 2), (12, 2), (7, 4), (13, 4), (14, 4)]
print(histogram([13, 7, 12, 7, 11, 13, 14, 13, 7, 11, 13, 14, 12, 14, 14, 7]))
```

```
[(11, 1), (7, 2), (12, 2), (14, 2), (13, 4)]
[(11, 2), (12, 2), (7, 4), (13, 4), (14, 4)]
```