

Python

Python and R for Data Science

Data Science and Management



Exercise 1: find how many equal numbers

Define a function `count_equals` that:

- takes as arguments four numbers
- returns:
 - the maximum number of equal numbers between the four

Example:

- `count_equals(1,2,3,4)` should return `0`
- `count_equals(1,2,5,4)` should return `0`
- `count_equals(1,2,2,2)` should return `3` because there are three `2` in the sequence
- `count_equals(1,1,1,2)` should return `3` because there are three `1` in the sequence

In [6]: *# Solution goes here*

Test your code

Run this code to test your solution:

```
In [7]: try: assert count_equals(1,2,3,4) == 0 and count_equals(1,5,3,4) == 0 and count_e  
except: print('Test failed')
```

Test failed

Exercise 2: Fibonacci's sequence

Define a function `fibonacci` that:

- takes as arguments:
 - an integer number `n`
- returns:
 - a list containing the first `n` numbers of the Fibonacci's sequence

Note: The Fibonacci sequence is a series of numbers where each number is the sum of the two previous ones, starting with 0 and 1. To calculate it, you begin with 0 and 1, then add these to get the next number. Continue this process to generate the sequence. It goes 0, 1, 1, 2, 3, 5, 8, and so on.

In [6]: *# Solution goes here*

Test your code

Run this code to test your solution:

```
In [7]: try: assert fibonacci(1) == [0] and fibonacci(3) == [0,1,1] and fibonacci(7) == [
except: print('Test failed')
```

Test failed

Exercise 3: zero-sum triplets

Define a function `zero_sum_triplets` that:

- takes as arguments:
 - a list of integers `numbers`
- returns:
 - the number of triplets whose sum is zero

Example:

- `zero_sum_triplets([1,-1,0,7,12])` should return `1` because the sum of 1,-1,0 is `0`
- `zero_sum_triplets([1,9,0,7,12])` should return `0` because there are no triplets that sum up to zero
- `zero_sum_triplets([1,-9,8,6,-14])` should return `2` because the sum of 1,-1,0 is `0` and the sum of 8,6,-14 is `0`

In [8]: *# Solution goes here*

Test your code

Run this code to test your solution:

```
In [9]: try: assert zero_sum_triplets([1,-1,0,7,12]) == 1 and zero_sum_triplets([1,9,0,7,  
except: print('Test failed')
```

Test passed

Exercise 4: Collatz

Define a function `collatz` that:

- takes as argument an integer number `n`
- returns:
 - a list containing all the numbers generated by the Collatz conjecture (stopping when reaching `1`)

Note: The Collatz Conjecture is a mathematical problem that starts with any positive integer. The process involves two steps: if the number is even, divide it by 2; if it's odd, multiply it by 3 and add 1. Repeat this process with the resulting number. The conjecture suggests that, no matter what number you start with, you'll eventually reach the number 1.

In [12]: *# Solution goes here*

Test your code

Run this code to test your solution:

```
In [13]: try: assert collatz(12) == [6,3,10,5,16,8,4,2,1] and collatz(1) == [] and collatz
except: print('Test failed')
```

Test failed

Exercise 5: Greatest Common Divisor (GCD)

Define a function `gcd` that:

- takes as argument two integer numbers `a` and `b`
- returns:
 - the gcd between `a` and `b`

In [15]: *# Solution goes here*

Test your code

Run this code to test your solution:

```
In [16]: try: assert gcd(1,2) == 1 and gcd(7,2) == 1 and gcd(4,2) == 2 and gcd(15,25) == 5
except: print('Test failed')
```

Test failed

Exercise 6: Factorial of a number

Define a function `factorial` that:

- takes as argument two integer numbers `a`
- returns:
 - the factorial of `a` (i.e., $n! = n * (n-1) * (n-2) * \dots * 1$)

```
In [ ]: # Solution goes here
```

Test your code

Run this code to test your solution:

```
In [ ]: try: assert factorial(1) == 1 and factorial(0) == 1 and factorial(5) == 120 and n
except: print('Test failed')
```

Test failed

Solution

```
In [9]: def factorial(x):
        result = 1
        for i in range(0,x):
            result *= x-i
        return result

# test
assert factorial(1) == 1 and factorial(0) == 1 and factorial(5) == 120 and not pr
```

120

Test passed

Exercise 7: Count vowels in a string

Define a function `vowels_counter` that:

- takes as argument a string `a`
- returns:
 - a dictionary with as key each vowel and as values the occurrences of each vowel

In []: *# Solution goes here*

Test your code

Run this code to test your solution:

```
In [ ]: try: assert vowels_counter("ciao") == {'i': 1, 'a': 1, 'o': 1} and vowels_counter
except: print('Test failed')
```

Test failed

Solution

```
In [13]: def vowels_counter(x):
          result = {}
          vowels = ["a", "e", "i", "o", "u"]
          for i in x:
              if i in vowels:
                  if i in result:
                      result[i] = result[i]+1
```

```
        else:
            result[i] = 1
    return result

# test
assert vowels_counter("ciao") == {'i': 1, 'a': 1, 'o': 1} and vowels_counter("abb
{'a': 2, 'e': 2, 'i': 1, 'o': 1}
Test passed
```

Exercise 8: Find missing number in a sequence

Define a function `find_missing` that:

- Takes a list of `n-1` integers, which represents a sequence of numbers from 1 to `n`, but one number is missing.
- Returns the missing number.

Note: Suppose that there is always only one number missing

Example:

- `find_missing([1, 2, 4, 5])` should return `3`.
- `find_missing([2, 3, 4, 6, 1])` should return `5`.

```
In [ ]: # Solution goes here
```

Test your code

Run this code to test your solution:

```
In [ ]: try: assert find_missing([1, 2, 4, 5]) == 3 and find_missing([2, 3, 4, 6, 1]) == 5
except: print('Test failed')
```

Test failed

Solution

```
In [17]: def find_missing(nums):
          n = len(nums) + 1
          total_sum = n * (n + 1) // 2
          actual_sum = sum(nums)

          return total_sum - actual_sum

# test
assert find_missing([1, 2, 4, 5]) == 3 and find_missing([2, 3, 4, 6, 1]) == 5 and
```

Test passed

Exercise 9: Longest Substring Without Repeating Characters

Define a function `longest_unique_substring` that:

- Takes a string as input.
- Returns the length of the longest substring that contains only unique characters.

Example:

- `longest_unique_substring("abcabcbb")` should return `3` (substring "abc").
- `longest_unique_substring("bbbbbb")` should return `1`.

```
In [ ]: # Solution goes here
```

Test your code

Run this code to test your solution:

```
In [ ]: try: assert longest_unique_substring("abcabcbb") == 3 and longest_unique_substring("bbbbbb") == 1
except: print('Test failed')
```


Test failed

Solution

```
In [16]: def longest_unique_substring(s):
max_length = 0

    for i in range(len(s)):
        seen_chars = set()

        for j in range(i, len(s)):
            if s[j] in seen_chars:
                break
            seen_chars.add(s[j])

        max_length = max(max_length, j - i)

    return max_length

# test
assert longest_unique_substring("abcabcbb") == 3 and longest_unique_substring("bb
```

Test passed

Exercise 10: Find the Majority Element

Define a function `majority_element` that:

- Takes a list of integers as input.
- Returns the element that appears more than half of the time in the list (if it exists). If no such element exists, return `None`.

Example:

- `majority_element([3, 3, 4, 2, 3, 3, 5])` should return `3`.
- `majority_element([1, 2, 3, 4, 5])` should return `None`.

```
In [ ]: # Solution goes here
```

Test your code

Run this code to test your solution:

```
In [ ]: try: assert majority_element([3, 3, 4, 2, 3, 3, 5]) == 3 and majority_element([1,
except: print('Test failed')
```

Test failed

Solution

```
In [14]: def majority_element(nums):
          count = {}
          n = len(nums)

          for num in nums:
              if num in count:
                  count[num] = count[num] + 1
              else:
                  count[num] = 1

              if count[num] > n // 2:
                  return num

          return None

          # test
          assert majority_element([3, 3, 4, 2, 3, 3, 5]) == 3 and majority_element([1, 2, 3
```

Test passed

