You

Please generate Python code that defines two integer variables and then does a simple addition operation.

Meta AI

Here is a simple Python code snippet that defines two integer variables and performs a simple addition operation:

Python

```
# Define two integer variables
num1 = 5
num2 = 3

# Perform addition operation
sum = num1 + num2

# Print the result
print("The sum is:", sum)
```

This code will output: `The sum is: 8`

Let me know if you have any other questions!

You

An isogram is a word that has no repeating letters, consecutive or non-consecutive. Implement a function that determines whether a string that contains only letters is an isogram. Assume the empty string is an isogram. Ignore letter case.

Meta AI

Here is a Python function that determines whether a string is an isogram:

Python

```
def is_isogram(string):
    """
    Determine if a string is an isogram (has no repeating letters).

    Args:
        string (str): The input string.

    Returns:
        bool: True if the string is an isogram, False otherwise.
    """
    # Convert the string to lowercase to ignore letter case
```

```
    string = string.lower()
    # Create a set from the string, which automatically removes duplicates
    unique_letters = set(string)
    # Check if the length of the set is equal to the length of the original s
    return len(unique_letters) == len(string)
```

Here's an explanation of the code:

1. We convert the input string to lowercase using the `lower()` method to ignore letter case.
2. We create a set from the string using the `set()` function, which automatically removes duplicates.
3. We compare the length of the set (`unique_letters`) with the length of the original string. If they are equal, it means there were no duplicates in the string, and we return `True` (the string is an isogram). Otherwise, we return `False`.

You can test the function with examples like this:

Python

```
print(is_isogram("abc"))  # True
print(is_isogram("aabb"))  # False
print(is_isogram("abcde"))  # True
print(is_isogram(""))  # True (empty string is considered an isogram)
```

I hope this helps! Let me know if you have any questions.

You

Check to see if a string has the same amount of 'x's and 'o's. The method must return a boolean and be case insensitive. The string can contain any char.

Meta AI

Here is a Python function that checks if a string has the same amount of 'x's and 'o's:

Python

```
def has_equal_x_and_o(string):
    """
    Check if a string has the same amount of 'x's and 'o's.

    Args:
        string (str): The input string.

    Returns:
        bool: True if the string has equal 'x's and 'o's, False otherwise.
    """
```

```
    # Convert the string to lowercase to ignore case
    string = string.lower()
    # Count the number of 'x's and 'o's
    x_count = string.count('x')
    o_count = string.count('o')
    # Return True if the counts are equal, False otherwise
    return x_count == o_count
```

Here's an explanation of the code:

1. We convert the input string to lowercase using the `lower()` method to ignore case.
2. We count the number of 'x's and 'o's in the string using the `count()` method.
3. We compare the counts using the == operator. If they are equal, we return `True` (the string has equal 'x's and 'o's). Otherwise, we return `False`.

You can test the function with examples like this:

Python

```
print(has_equal_x_and_o("xo"))  # True
print(has_equal_x_and_o("xxoo"))  # True
print(has_equal_x_and_o("xoXO"))  # True (case insensitive)
print(has_equal_x_and_o("xxxoo"))  # False
print(has_equal_x_and_o("xOoX"))  # True
```

I hope this helps! Let me know if you have any questions.

You

A pangram is a sentence that contains every single letter of the alphabet at least once. For example, the sentence "The quick brown fox jumps over the lazy dog" is a pangram, because it uses the letters A-Z at least once (case is irrelevant).
Given a string, detect whether or not it is a pangram. Return True if it is, False if not. Ignore numbers and punctuation

Meta AI

Here is a Python function that detects whether a string is a pangram:

Python

```
def is_pangram(string):
    """
    Detect whether a string is a pangram (contains every letter of the alphab

    Args:
        string (str): The input string.
```

```
    Returns:
        bool: True if the string is a pangram, False otherwise.
    """
    # Convert the string to lowercase and remove non-alphabetic characters
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    string = "".join(filter(str.isalpha, string.lower()))
    # Check if every letter of the alphabet is in the string
    return set(alphabet) <= set(string)
```

Here's an explanation of the code:

1. We define the `alphabet` string containing all lowercase letters.
2. We convert the input string to lowercase using the `lower()` method and remove non-alphabetic characters using the `filter()` function and `str.isalpha` predicate.
3. We create a set from the filtered string, which automatically removes duplicates.
4. We check if the set of alphabet letters is a subset of the set of string letters using the `<=` operator. If it is, we return `True` (the string is a pangram). Otherwise, we return `False`.

You can test the function with examples like this:

Python

```
print(is_pangram("The quick brown fox jumps over the lazy dog"))  # True
print(is_pangram("Hello, World!"))  # False
print(is_pangram("The five boxing wizards jump quickly at dawn."))  # True
print(is_pangram("This sentence is not a pangram."))  # False
```

I hope this helps! Let me know if you have any questions.

You

Write a function, persistence, that takes in a positive parameter num and returns its multiplicative persistence, which is the number of times you must multiply the digits in num until you reach a single digit.

Meta AI

Here is a Python function that calculates the multiplicative persistence of a number:

Python

```
def persistence(num):
    """
    Calculate the multiplicative persistence of a number.

    Args:
        num (int): A positive integer.
```

```
    Returns:
        int: The multiplicative persistence of the number.
    """
    # Convert the number to a string to easily access its digits
    num_str = str(num)
    # Initialize the persistence counter
    persistence_count = 0
    # Loop until the number is reduced to a single digit
    while len(num_str) > 1:
        # Initialize the product of the digits
        product = 1
        # Calculate the product of the digits
        for digit in num_str:
            product *= int(digit)
        # Update the number string and increment the persistence counter
        num_str = str(product)
        persistence_count += 1
    # Return the persistence count
    return persistence_count
```

Here's an explanation of the code:

1. We convert the input number to a string to easily access its digits.
2. We initialize the persistence counter to 0.
3. We loop until the number is reduced to a single digit.
4. Inside the loop, we calculate the product of the digits and update the number string.
5. We increment the persistence counter at each iteration.
6. Finally, we return the persistence count.

You can test the function with examples like this:

Python

```
print(persistence(39))  # 3
print(persistence(999))  # 4
print(persistence(4))  # 0
```

I hope this helps! Let me know if you have any questions.

You

Write a function cakes(), which takes the recipe (object) and the available ingredients (also an object) and returns the maximum number of cakes Pete can bake (integer). For simplicity there are no units for the amounts (e.g. 1 lb of flour or 200 g of sugar are simply 1 or 200). Ingredients that are not present in the objects, can be considered as 0.

Meta AI

Here is a Python function that calculates the maximum number of cakes that can be baked:

Python

```
def cakes(recipe, available):
    """
    Calculate the maximum number of cakes that can be baked.

    Args:
        recipe (dict): A dictionary containing the recipe (e.g. {"flour": 500
        available (dict): A dictionary containing the available ingredients (

    Returns:
        int: The maximum number of cakes that can be baked.
    """
    # Initialize the maximum number of cakes to infinity
    max_cakes = float('inf')
    # Iterate over the ingredients in the recipe
    for ingredient, amount in recipe.items():
        # Get the available amount of the ingredient (default to 0 if not pre
        available_amount = available.get(ingredient, 0)
        # Calculate the number of cakes that can be baked with this ingredier
        cakes_with_ingredient = available_amount // amount
        # Update the maximum number of cakes
        max_cakes = min(max_cakes, cakes_with_ingredient)
    # Return the maximum number of cakes
    return max_cakes
```

Here's an explanation of the code:

1. We initialize the maximum number of cakes to infinity.
2. We iterate over the ingredients in the recipe.
3. For each ingredient, we get the available amount (defaulting to 0 if not present).
4. We calculate the number of cakes that can be baked with this ingredient by dividing the available amount by the recipe amount.
5. We update the maximum number of cakes by taking the minimum of the current maximum and the number of cakes with this ingredient.
6. Finally, we return the maximum number of cakes.

You can test the function with examples like this:

Python

```
recipe = {"flour": 500, "sugar": 200, "eggs": 1}
available = {"flour": 1200, "sugar": 1200, "eggs": 5, "milk": 3000}
print(cakes(recipe, available))  # 2
```

I hope this helps! Let me know if you have any questions.

You

You probably know the "like" system from Facebook and other pages. People can "like" blog posts, pictures or other items. We want to create the text that should be displayed next to such an item. Implement the function which takes an array containing the names of people that like an item. It must return the display text as shown in the examples

Meta AI

Here is a Python function that generates the display text based on the list of names:

Python

```
def likes(names):
    """
    Generate the display text for the "like" system.

    Args:
        names (list): A list of names of people who like the item.

    Returns:
        str: The display text.
    """
    if len(names) == 0:
        return "no one likes this"
    elif len(names) == 1:
        return "{} likes this".format(names[0])
    elif len(names) == 2:
        return "{} and {} like this".format(names[0], names[1])
    elif len(names) == 3:
        return "{}, {} and {} like this".format(names[0], names[1], names[2])
    else:
        return "{}, {} and {} others like this".format(names[0], names[1], le
```

Here's an explanation of the code:

1. If the list of names is empty, return "no one likes this".
2. If there is only one name, return "{} likes this" with the name inserted.
3. If there are two names, return "{} and {} like this" with the names inserted.
4. If there are three names, return "{}, {} and {} like this" with the names inserted.

5. If there are more than three names, return "{}, {} and {} others like this" with the first two names and the number of additional names inserted.

You can test the function with examples like this:

Python

```
print(likes([]))  # "no one likes this"
print(likes(["Peter"]))  # "Peter likes this"
print(likes(["Jacob", "Alex"]))  # "Jacob and Alex like this"
print(likes(["Max", "John", "Mark"]))  # "Max, John and Mark like this"
print(likes(["Alex", "Jacob", "Mark", "Max"]))  # "Alex, Jacob and 2 others l
```

I hope this helps! Let me know if you have any questions.

You

Digital root is the recursive sum of all the digits in a number.
Given n, take the sum of the digits of n. If that value has more than one digit, continue reducing in this way until a single-digit number is produced. The input will be a non-negative integer.

Meta AI

Here is a Python function that calculates the digital root of a number:

Python

```
def digital_root(n):
    """
    Calculate the digital root of a number.

    Args:
        n (int): A non-negative integer.

    Returns:
        int: The digital root of the number.
    """
    while n > 9:
        n = sum(int(digit) for digit in str(n))
    return n
```

Here's an explanation of the code:

1. We use a while loop to repeatedly calculate the sum of the digits until we reach a single-digit number.
2. Inside the loop, we convert the number to a string to easily access its digits.
3. We use a generator expression to sum the digits, converting each digit back to an integer.

4. We assign the sum back to the variable n.

5. Once we reach a single-digit number, we return it as the digital root.

You can test the function with examples like this:

Python

```
print(digital_root(16))  # 7
print(digital_root(456))  # 6
print(digital_root(942))  # 6
```

I hope this helps! Let me know if you have any questions.

You

Your task is to sort a given string. Each word in the string will contain a single number. This number is the position the word should have in the result.
Note: Numbers can be from 1 to 9. So 1 will be the first word (not 0).
If the input string is empty, return an empty string. The words in the input String will only contain valid consecutive numbers.

Meta AI

Here is a Python function that sorts the string based on the numbers in each word:

```
def order_words(s):
    """
    Sort a string based on the numbers in each word.

    Args:
        s (str): The input string.

    Returns:
        str: The sorted string.
    """
    if not s:
        return ""
    words = s.split()
    sorted_words = [""] * len(words)
    for word in words:
        for char in word:
            if char.isdigit():
                sorted_words[int(char) - 1] = word
                break
    return " ".join(sorted_words)
```

Here's an explanation of the code:

1. If the input string is empty, return an empty string.
2. Split the input string into words.
3. Create a list of empty strings with the same length as the number of words.
4. Iterate over each word in the input string.
5. Find the digit in the word and use it as the index (minus 1) to place the word in the sorted list.
6. Join the sorted list of words into a single string with spaces in between.

You can test the function with examples like this:

Python

```
print(order_words("is2 Thi1 a3 Test4"))  # "Thi1 is2 a3 Test4"
print(order_words("4of Fo1r pe6ople g3ood th5e the2"))  # "Fo1r the2 g3ood 4c
print(order_words(""))  # ""
```

I hope this helps! Let me know if you have any questions.

You

Given an array of ones and zeroes, convert the equivalent binary value to an integer.
Eg: [0, 0, 0, 1] is treated as 0001 which is the binary representation of 1.

Meta AI

Here is a Python function that converts an array of binary digits to an integer:

Python

```
def binary_array_to_number(arr):
    """
    Convert an array of binary digits to an integer.

    Args:
        arr (list): A list of binary digits (0s and 1s).

    Returns:
        int: The equivalent integer value.
    """
    return int("".join(map(str, arr)), 2)
```

Here's an explanation of the code:

1. Use the `map` function to convert each binary digit to a string.
2. Join the strings together using `"".join()`.

3. Convert the resulting binary string to an integer using `int()` with base 2.

You can test the function with examples like this:

Python

```
print(binary_array_to_number([0, 0, 0, 1]))  # 1
print(binary_array_to_number([1, 1, 1, 1]))  # 15
print(binary_array_to_number([0, 1, 0, 0]))  # 4
```

I hope this helps! Let me know if you have any questions.

You

The rgb function is incomplete. Complete it so that passing in RGB decimal values will result in a hexadecimal representation being returned. Valid decimal values for RGB are 0 - 255. Any values that fall out of that range must be rounded to the closest valid value.

Note: Your answer should always be 6 characters long, the shorthand with 3 will not work here.

Meta AI

Here is the completed rgb function:

Python

```
def rgb(r, g, b):
    """
    Convert RGB decimal values to a hexadecimal representation.

    Args:
        r (int): The red value (0-255).
        g (int): The green value (0-255).
        b (int): The blue value (0-255).

    Returns:
        str: The hexadecimal representation of the RGB values.
    """
    # Ensure values are within the valid range (0-255)
    r = min(max(r, 0), 255)
    g = min(max(g, 0), 255)
    b = min(max(b, 0), 255)

    # Convert decimal values to hexadecimal
    hex_r = "{:02x}".format(r)
    hex_g = "{:02x}".format(g)
    hex_b = "{:02x}".format(b)
```

```
    # Return the hexadecimal representation
    return "#" + hex_r + hex_g + hex_b
```

Here's an explanation of the code:

1. Ensure the input values are within the valid range (0-255) using `min` and `max`.
2. Convert each decimal value to a two-digit hexadecimal representation using `"{:02x}".format()`.
3. Return the hexadecimal representation in the format `#rrggbb`.

You can test the function with examples like this:

Python

```
print(rgb(255, 255, 255))  # "#ffffff"
print(rgb(0, 0, 0))  # "#000000"
print(rgb(148, 0, 211))  # "#9400d3"
```

I hope this helps! Let me know if you have any questions.

You

Write an algorithm that takes an array and moves all of the zeros to the end, preserving the order of the other elements.

Meta AI

Here is a Python function that moves all zeros to the end of an array:

Python

```
def move_zeros_to_end(arr):
    """
    Move all zeros to the end of an array, preserving the order of other elem

    Args:
        arr (list): The input array.

    Returns:
        list: The modified array with zeros moved to the end.
    """
    # Separate the array into two lists: non-zeros and zeros
    non_zeros = [x for x in arr if x != 0]
    zeros = [x for x in arr if x == 0]

    # Combine the non-zeros and zeros lists
    return non_zeros + zeros
```

Here's an explanation of the code:

1. Use list comprehensions to separate the array into two lists: non_zeros (elements that are not zero) and zeros (elements that are zero).
2. Combine the non_zeros and zeros lists using the + operator.

You can test the function with examples like this:

Python

```
print(move_zeros_to_end([0, 1, 0, 3, 12]))  # [1, 3, 12, 0, 0]
print(move_zeros_to_end([1, 2, 0, 4, 0, 5]))  # [1, 2, 4, 5, 0, 0]
```

I hope this helps! Let me know if you have any questions.

You

The goal of this exercise is to convert a string to a new string where each character in the new string is "(" if that character appears only once in the original string, or ")" if that character appears more than once in the original string. Ignore capitalization when determining if a character is a duplicate.

Meta AI

Here is a Python function that achieves the goal:

Python

```python
def duplicate_encode(word):
    """
    Convert a string to a new string where each character is "(" if it appear
    or ")" if it appears more than once.

    Args:
        word (str): The input string.

    Returns:
        str: The modified string with "(" and ")" characters.
    """
    word = word.lower()
    count_dict = {}
    for char in word:
        if char in count_dict:
            count_dict[char] += 1
        else:
            count_dict[char] = 1
    result = ""
    for char in word:
        if count_dict[char] == 1:
            result += "("
        else:
            result += ")"
    return result
```

Here's an explanation of the code:

1. Convert the input string to lowercase using `word.lower()`.
2. Create a dictionary `count_dict` to store the count of each character.
3. Iterate over the characters in the string and update the count in the dictionary.
4. Iterate over the characters in the string again and build the result string using "(" for single occurrences and ")" for duplicates.
5. Return the modified string.

You can test the function with examples like this:

Python

```python
print(duplicate_encode("Hello World"))  # "((((()))))()("
print(duplicate_encode("abcde"))  # "(((((("
```

I hope this helps! Let me know if you have any questions.

You

Write a method that takes an array of consecutive (increasing) letters as input and that returns the missing letter in the array.
You will always get an valid array. And it will be always exactly one letter be missing. The length of the array will always be at least 2.
The array will always contain letters in only one case.

Meta AI

Here is a Python function that finds the missing letter in the array:

Python

```python
def find_missing_letter(chars):
    """
    Find the missing letter in an array of consecutive letters.

    Args:
        chars (list): The input array of consecutive letters.

    Returns:
        str: The missing letter.
    """
    for i in range(len(chars) - 1):
        if ord(chars[i + 1]) - ord(chars[i]) > 1:
            return chr(ord(chars[i]) + 1)
```

Here's an explanation of the code:

1. Iterate over the array of letters.
2. Check if the difference between the ASCII values of the current and next letters is greater than 1.
3. If it is, return the missing letter by adding 1 to the ASCII value of the current letter using `chr` and `ord`.
4. If no missing letter is found, the function will implicitly return `None`.

You can test the function with examples like this:

Python

```python
print(find_missing_letter(['a', 'b', 'd', 'e']))  # 'c'
print(find_missing_letter(['O', 'Q', 'R', 'S']))  # 'P'
```

I hope this helps! Let me know if you have any questions.

You

Given an n x n array, return the array elements arranged from outermost elements to the middle element, traveling clockwise.

Meta AI

Here is a Python function that arranges the array elements from outermost to the middle element, traveling clockwise:

```
def arrange_clockwise(arr):
    """
    Arrange the array elements from outermost to the middle element, travelir

    Args:
        arr (list): The input 2D array.

    Returns:
        list: The arranged array elements.
    """
    result = []
    while arr:
        # Append the first row
        result += arr.pop(0)
        # Append the last column
        if arr and arr[0]:
            for row in arr:
                result.append(row.pop())
        # Append the last row in reverse order
        if arr:
            result += arr.pop()[::-1]
        # Append the first column in reverse order
        if arr and arr[0]:
            for row in arr[::-1]:
                result.append(row.pop(0))
    return result
```

Here's an explanation of the code:

1. Initialize an empty list `result` to store the arranged elements.
2. Loop until the input array is empty.
3. Append the first row of the array to `result` and remove it from the array.
4. If the array is not empty, append the last column of the array to `result` and remove it from the array.

5. If the array is not empty, append the last row of the array in reverse order to `result` and remove it from the array.
6. If the array is not empty, append the first column of the array in reverse order to `result` and remove it from the array.
7. Return the arranged array elements in `result`.

You can test the function with examples like this:

Python

```
arr = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(arrange_clockwise(arr))  # [1, 2, 3, 6, 9, 8, 7, 4, 5]
```

I hope this helps! Let me know if you have any questions.

You

Define a function that takes in two non-negative integers ﷿a and ﷿b and returns the last decimal digit of ﷿﷿ab. Note that ﷿a and ﷿b may be very large!
For example, the last decimal digit of 9797 is 99, since 97=478296997=4782969. The last decimal digit of (2200)2300(2200)2300, which has over 10921092 decimal digits, is 66. Also, please take 0000 to be 11.
You may assume that the input will always be valid.

Meta AI

Here is a Python function that returns the last decimal digit of a^b:

Python

```
def last_decimal_digit(a, b):
    """
    Return the last decimal digit of a^b.

    Args:
        a (int): The base number.
        b (int): The exponent.

    Returns:
        int: The last decimal digit of a^b.
    """
    # Calculate the last decimal digit using modular arithmetic
    return pow(a, b, 10)
```

Here's an explanation of the code:

1. Use the `pow` function with three arguments: `a`, `b`, and `10`.

2. The `pow` function calculates `a` raised to the power of `b` modulo `10`, which gives the last decimal digit.

You can test the function with examples like this:

Python

```
print(last_decimal_digit(97, 2))  # 9
print(last_decimal_digit(2200, 2300))  # 6
```

I hope this helps! Let me know if you have any questions.

You

Write a function that when given a URL as a string, parses out just the domain name and returns it as a string.

Meta AI

Here is a Python function that extracts the domain name from a URL:

```
import urllib.parse

def extract_domain(url):
    """
    Extract the domain name from a URL.

    Args:
        url (str): The input URL.

    Returns:
        str: The domain name.
    """
    parsed_uri = urllib.parse.urlparse(url)
    domain = '{uri.netloc}'.format(uri=parsed_uri)
    return domain
```

Here's an explanation of the code:

1. Import the `urllib.parse` module.
2. Use the `urlparse` function to parse the input URL.
3. Extract the `netloc` attribute, which contains the domain name.
4. Return the domain name as a string.

You can test the function with examples like this:

Python

```
print(extract_domain("(link unavailable)"))  # "(link unavailable)"
print(extract_domain("(link unavailable)"))  # "(link unavailable)"
```

I hope this helps! Let me know if you have any questions.

You

Write two functions that convert a roman numeral to and from an integer value. Multiple roman numeral values will be tested for each function.
Modern Roman numerals are written by expressing each digit separately starting with the left most digit and skipping any digit with a value of zero. In Roman numerals:
1990 is rendered: 1000=M, 900=CM, 90=XC; resulting in MCMXC
2008 is written as 2000=MM, 8=VIII; or MMVIII
1666 uses each Roman symbol in descending order: MDCLXVI.
Input range : 1 <= n < 4000
In this kata 4 should be represented as IV, NOT as IIII (the "watchmaker's four").

Meta AI

Here are the two functions that convert a Roman numeral to and from an integer value:

```
def roman_to_int(s):
    roman_numerals = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, '
    int_val = 0
    for i in range(len(s)):
        if i > 0 and roman_numerals[s[i]] > roman_numerals[s[i - 1]]:
            int_val += roman_numerals[s[i]] - 2 * roman_numerals[s[i - 1]]
        else:
            int_val += roman_numerals[s[i]]
    return int_val

def int_to_roman(num):
    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4,
        1
        ]
    syb = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
        ]
```

```
    roman_num = ''
    i = 0
    while  num > 0:
        for _ in range(num // val[i]):
            roman_num += syb[i]
            num -= val[i]
        i += 1
    return roman_num
```

Here's an explanation of the code:

roman_to_int(s)

1. Define a dictionary `roman_numerals` that maps Roman numerals to their integer values.
2. Initialize an integer `int_val` to 0.
3. Iterate over the input string `s`.
4. If the current numeral is greater than the previous one, subtract twice the value of the previous numeral (to account for the subtraction rule in Roman numerals).
5. Add the value of the current numeral to `int_val`.
6. Return `int_val`.

int_to_roman(num)

1. Define two lists `val` and `syb` that contain the integer values and corresponding Roman numerals, respectively.
2. Initialize an empty string `roman_num`.
3. Loop until `num` is 0.
4. Add the corresponding Roman numeral to `roman_num` for each iteration of the loop.
5. Subtract the value of the current numeral from `num`.
6. Return `roman_num`.

You can test the functions with examples like this:

Python

```python
print(roman_to_int('III'))  # 3
print(roman_to_int('IV'))  # 4
print(roman_to_int('IX'))  # 9
print(roman_to_int('LVIII'))  # 58
print(roman_to_int('MCMXCIV'))  # 1994

print(int_to_roman(4))  # IV
print(int_to_roman(9))  # IX
print(int_to_roman(13))  # XIII
```

```
print(int_to_roman(44))  # XLIV
print(int_to_roman(1000))  # M
```

I hope this helps! Let me know if you have any questions.

You

In this kata we want to convert a string into an integer. The strings simply represent the numbers in words.
Examples:
"one" => 1
"twenty" => 20
"two hundred forty-six" => 246
"seven hundred eighty-three thousand nine hundred and nineteen" => 783919
Additional Notes:
The minimum number is "zero" (inclusively)
The maximum number, which must be supported is 1 million (inclusively)
The "and" in e.g. "one hundred and twenty-four" is optional, in some cases it's present and in others it's not
All tested numbers are valid, you don't need to validate them

Meta AI

Here is a Python function that converts a string representing a number in words to an integer:

```
def word_to_num(s):
    num_dict = {
        "zero": 0, "one": 1, "two": 2, "three": 3, "four": 4, "five": 5,
        "six": 6, "seven": 7, "eight": 8, "nine": 9, "ten": 10,
        "eleven": 11, "twelve": 12, "thirteen": 13, "fourteen": 14,
        "fifteen": 15, "sixteen": 16, "seventeen": 17, "eighteen": 18,
        "nineteen": 19, "twenty": 20, "thirty": 30, "forty": 40,
        "fifty": 50, "sixty": 60, "seventy": 70, "eighty": 80,
        "ninety": 90, "hundred": 100, "thousand": 1000, "million": 1000000
    }
    s = s.replace("-", " ")
    s = s.replace(" and ", " ")
    words = s.split()
    num = 0
    temp = 0
    for word in words:
        if word in num_dict:
            if word == "hundred":
                temp *= num_dict[word]
            elif word == "thousand" or word == "million":
                num += temp * num_dict[word]
                temp = 0
```

```
            else:
                temp += num_dict[word]
    num += temp
    return num
```

Here's an explanation of the code:

1. Define a dictionary `num_dict` that maps number words to their integer values.
2. Replace hyphens and " and " with spaces in the input string.
3. Split the string into words.
4. Initialize two variables: `num` to store the final result and `temp` to store the temporary result.
5. Iterate over the words.
6. If the word is in `num_dict`, perform the following operations:
    - If the word is "hundred", multiply `temp` by 100.
    - If the word is "thousand" or "million", add `temp` multiplied by the value of the word to `num` and reset `temp` to 0.
    - Otherwise, add the value of the word to `temp`.
7. Add `temp` to `num` after the loop.
8. Return `num`.

You can test the function with examples like this:

Python

```
print(word_to_num("one"))  # 1
print(word_to_num("twenty"))  # 20
print(word_to_num("two hundred forty-six"))  # 246
print(word_to_num("seven hundred eighty-three thousand nine hundred and ninet
```

I hope this helps! Let me know if you have any questions.

You

Your task in order to complete this Kata is to write a function which formats a duration, given as a number of seconds, in a human-friendly way.
The function must accept a non-negative integer. If it is zero, it just returns "now". Otherwise, the duration is expressed as a combination of years, days, hours, minutes and seconds.
It is much easier to understand with an example:
* For seconds = 62, your function should return
"1 minute and 2 seconds"
* For seconds = 3662, your function should return
"1 hour, 1 minute and 2 seconds"
For the purpose of this Kata, a year is 365 days and a day is 24 hours.
Note that spaces are important.

Detailed rules
The resulting expression is made of components like 4 seconds, 1 year, etc. In general, a positive integer and one of the valid units of time, separated by a space. The unit of time is used in plural if the integer is greater than 1.
The components are separated by a comma and a space (", "). Except the last component, which is separated by " and ", just like it would be written in English.
A more significant units of time will occur before than a least significant one. Therefore, 1 second and 1 year is not correct, but 1 year and 1 second is.
Different components have different unit of times. So there is not repeated units like in 5 seconds and 1 second.
A component will not appear at all if its value happens to be zero. Hence, 1 minute and 0 seconds is not valid, but it should be just 1 minute.
A unit of time must be used "as much as possible". It means that the function should not return 61 seconds, but 1 minute and 1 second instead. Formally, the duration specified by of a component must not be greater than any valid more significant unit of time.

Meta AI

Here is a Python function that formats a duration in a human-friendly way:

```
def format_duration(seconds):
    if seconds == 0:
        return "now"
    components = []
    years, seconds = divmod(seconds, 31536000)
    if years > 0:
        components.append("{} year{}".format(years, "" if years == 1 else "s"
    days, seconds = divmod(seconds, 86400)
    if days > 0:
        components.append("{} day{}".format(days, "" if days == 1 else "s"))
    hours, seconds = divmod(seconds, 3600)
    if hours > 0:
        components.append("{} hour{}".format(hours, "" if hours == 1 else "s"
    minutes, seconds = divmod(seconds, 60)
    if minutes > 0:
        components.append("{} minute{}".format(minutes, "" if minutes == 1 el
    if seconds > 0:
        components.append("{} second{}".format(seconds, "" if seconds == 1 el
    if len(components) == 1:
        return components[0]
    last_component = components.pop()
    return ", ".join(components) + " and " + last_component
```

Here's an explanation of the code:

1. If the input `seconds` is 0, return "now".
2. Initialize an empty list `components` to store the components of the duration.
3. Calculate the years, days, hours, minutes, and seconds using divmod.
4. If a component is greater than 0, add it to the `components` list.
5. If there is only one component, return it.
6. Otherwise, join all components except the last one with ", " and add the last component with " and ".

You can test the function with examples like this:

Python

```
print(format_duration(62))  # "1 minute and 2 seconds"
print(format_duration(3662))  # "1 hour, 1 minute and 2 seconds"
```

I hope this helps! Let me know if you have any questions.