100 / 100 pts

Group_06_Project

February 24, 2021

1 Mini-projects

1.1 Help Function for tests

Test your result with the `assert_equals` function. Do not forget to compile it first otherwise you will get a

```
NameError: name 'assert_equals' is not defined
```

[1]:
```
def assert_equals(a, b):
    return a == b
```

1.2 Project 1

IGNORED

Have you ever felt like there was something more being said behind a paragraph, a sentence? In this project you have to find all duplicates from our sentence to uncover the real message.

Your job is to write a function that will find the secret words of the message and return them in order. The words in the secret message should be ordered in the order in which they are found as a duplicate, for example:

`'This is a test. this test is fun.' should return 'this test is'`

. Notes

The input will always be a string.

The ponctuation like “.” (point), “!” (exclamation mark), “?” (question mark), “:” (colon) after words should be removed before the check.

If a word appears more than two times in the paragraph, it should show up in the secret message only once, based on the position of the first time it was duplicated.

The punctuation and casing of words (uppercase, lowercase) should not matter for the purpose of this project. We are only concerned with word duplication.

[5]:
```
import re

paragraph = "This is a test. this test is fun."
expected_result = "this test is"

def find_secret_message(paragraph):

```

```
    assert_equals = re.compile(r".*(.).*\1")
    expected_result = assert_equals.findall(paragraph)

print(assert_equals(find_secret_message(paragraph), expected_result))
```

```
False
```

[ ]:
```
# Basic Tests
paragraph = "This is a test. this test is fun."
expected_result = "this test is"
print(assert_equals(find_secret_message(paragraph), expected_result))
################################################################################
paragraph = "There is a secret message in the first six sentences of this kata␣
 ↪description. Have you ever felt like there was something more being said?␣
 ↪Was it hard to figure out that unspoken meaning? Never again! Never will a␣
 ↪secret go undiscovered. Find all duplicates from our message!"
expected_result = "there was never a secret message"
print(assert_equals(find_secret_message(paragraph), expected_result))
################################################################################
paragraph = "think T-rex! code, T-Rex chocolate wants kills? wants T-Rex secret:
 ↪ sleeps chocolate talks not! good:"
expected_result = "t-rex wants chocolate"
print(assert_equals(find_secret_message(paragraph), expected_result))
################################################################################
paragraph = "Even me as a someone who hates devil was really upset to an angel␣
 ↪once. Even if the devil was my an bad angel once we got friend."
expected_result = "even devil was an angel once"
print(assert_equals(find_secret_message(paragraph), expected_result))
################################################################################
paragraph = "Christmas is not! Christmas season it is not a. a is . season It's␣
 ↪feeling it's feeling."
expected_result = "christmas is not a season it's feeling"
print(assert_equals(find_secret_message(paragraph), expected_result))
```

1.3 Project 2

50 / 50 pts

A person has a rather old digital-piano being worth 2000€. He saw a secondhand digital-piano being worth 8000€.

He wants to keep his old digital-piano until he can buy the secondhand one.

He know he can save 1000€ each month but the prices of his old digital-piano and of the new one decrease of 1.5 percent per month. Furthermore this percent of loss increases of 0.5 percent at the end of every two months.

He find it too difficult to make all these calculations.

Your task is to help him.

How many months will it take him to save up enough money to buy the digital-piano he wants, and how much money will he have left over?

Parameters and return of function:

```
start_price_old (old digital-piano price): (positive int or float, guaranteed)
start_price_new (new digital-piano price): (positive int or float, guaranteed)
saving_per_month: (positive int or float, guaranteed)
percent_loss_by_bonth: (int or float)
number_month(2000, 8000, 1000, 1.5) should return [6, 766]
```

Detail of the above example:

```
end month 1: percentLoss 1.5 available -4910.0
end month 2: percentLoss 2.0 available -3791.7999...
end month 3: percentLoss 2.0 available -2675.964
end month 4: percentLoss 2.5 available -1534.06489...
end month 5: percentLoss 2.5 available -395.71327...
end month 6: percentLoss 3.0 available 766.158120825...

return [6, 766]
```

where 6 is the number of months at the end of which he can buy the new used digital-piano and 766 is the nearest integer to 766.158… (rounding 766.158 gives 766).

Note:

Selling, buying and saving are normally done at end of month. Calculations are processed at the end of each considered month but if, by chance from the start, the value of the old digital-piano is bigger than the value of the new one or equal there is no saving to be made, no need to wait so he can at the beginning of the month buy the new used digita-piano:

```
number_month(12000, 8000, 1000, 1.5) should return [0, 4000]
number_month(8000, 8000, 1000, 1.5) should return [0, 0]
```

[9]:
```python
import math # in case it's needed

def number_month(start_price_old, start_price_new, saving_per_month,␣
↪percent_loss_by_month):
    number_month = 0
    monthly_saved_money = 0
    while start_price_old + monthly_saved_money < start_price_new:
        monthly_saved_money += saving_per_month
        number_month += 1
        if number_month % 2 == 0:
            percent_loss_by_month += 0.5
        start_price_old *= ((100 - percent_loss_by_month) / 100)
        start_price_new *= ((100 - percent_loss_by_month) / 100)

    return [number_month, round(start_price_old + monthly_saved_money -␣
↪start_price_new)]
```

```
#The round() function returns a floating point number that is a rounded version␣
 ↪of the specified number, with the specified number of decimals.

#The default number of decimals is 0, meaning that the function will return the␣
 ↪nearest integer.
```

[12]:
```
number_month(5320, 15320, 1000, 1.4)
```

[12]:
```
[9, 1047]
```

[15]:
```
# Basic Tests
start_price_old = 2000
start_price_new = 8000
saving_per_month = 1000
percent_loss_by_month = 1.5
expected_result = [6, 766]
print(assert_equals(number_month(start_price_old, start_price_new,␣
 ↪saving_per_month, percent_loss_by_month), expected_result))
################################################################################
start_price_old = 12000
start_price_new = 8000
saving_per_month = 1000
percent_loss_by_month = 1.5
expected_result = [0, 4000]
print(assert_equals(number_month(start_price_old, start_price_new,␣
 ↪saving_per_month, percent_loss_by_month), expected_result))
################################################################################
start_price_old = 2100
start_price_new = 2100
saving_per_month = 1000
percent_loss_by_month = 1.2
expected_result = [0, 0]
print(assert_equals(number_month(start_price_old, start_price_new,␣
 ↪saving_per_month, percent_loss_by_month), expected_result))
################################################################################
start_price_old = 7500
start_price_new = 32000
saving_per_month = 300
percent_loss_by_month = 1.55
expected_result = [25, 122]
print(assert_equals(number_month(start_price_old, start_price_new,␣
 ↪saving_per_month, percent_loss_by_month), expected_result))
################################################################################
start_price_old = 5320
start_price_new = 15320
saving_per_month = 1000
```

```
percent_loss_by_month = 1.4
expected_result = [9, 1047]
print(assert_equals(number_month(start_price_old, start_price_new,␣
 ↪saving_per_month, percent_loss_by_month), expected_result))
```

```
True
True
True
True
True
```

50 / 50 pts

1.4 Project 3

A store manager has lots of products classified in 26 categories labeled A, B, … Z. Each product has a code `c` of 3, 4, 5 or more capitals letters. The 1st letter of a code is the capital letter of the product category. In the store manager's list each code `c` is followed by a space and by a positive integer `n` (int n >= 0) which indicates the quantity of products of this code in stock.

For example an extract of one of the stocklists could be:

`L = ["ABART 20", "CDXEF 50", "BKWRK 25", "BTSQZ 89", "DRTYM 60"]` `

You will be given a stocklist (e.g. : L) and a list of categories in capital letters e.g :

`M = ["A", "B", "C", "W"]`

Your task is to find all the products of `L` with codes belonging to each category of `M` and to sum their quantity according to each category.

For the lists L and M of example you have to return the string:

"(A : 20) - (B : 114) - (C : 50) - (W : 0)"

where **A, B, C, W** are the categories, 20 is the sum of the unique product of category A, 114 the sum corresponding to "BKWRK" and "BTSQZ", 50 corresponding to "CDXEF" and 0 to category 'W' since there are no code beginning with W.

If `L` or `M` are empty return string is

""

Note: In the result codes and their values are in the same order as in M.

[18]:
```
def stock_list(list_of_prod, list_of_cat):
    if len(list_of_cat) == 0 or len(list_of_prod) == 0:
        return ""
    answer = ""
    for letter in list_of_cat:
        summe = 0
        for product in list_of_prod:
            if product[0] == letter[0]:
                summe += int(product.split(" ")[1])
        if len(answer) != 0:
```

```
            answer += " - "
        answer += "(" + str(letter) + " : " + str(summe) + ")"
    return answer
```

[13]:
```
# Basic Tests
list_of_prod = ["ABAR 200", "CDXE 500", "BKWR 250", "BTSQ 890", "DRTY 600"]
list_of_cat = ["A", "B"]
expected_result = "(A : 200) - (B : 1140)"
print(assert_equals(stock_list(list_of_prod, list_of_cat), expected_result))
################################################################################
list_of_prod = ['BBAR 150', 'CDXE 515', 'BKWR 250', 'BTSQ 890', 'DRTY 600']
list_of_cat = ['A', 'B', 'C', 'D']
expected_result = "(A : 0) - (B : 1290) - (C : 515) - (D : 600)"
print(assert_equals(stock_list(list_of_prod, list_of_cat), expected_result))
################################################################################
list_of_prod = ['ABAR 200', 'CDXE 500', 'BKWR 250', 'BTSQ 890', 'DRTY 600']
list_of_cat = ['A', 'B']
expected_result = "(A : 200) - (B : 1140)"
print(assert_equals(stock_list(list_of_prod, list_of_cat), expected_result))
################################################################################
list_of_prod = ['CBART 20', 'CDXEF 50', 'BKWRK 25', 'BTSQZ 89', 'DRTYM 60']
list_of_cat = ['A', 'B', 'C', 'W']
expected_result = "(A : 0) - (B : 114) - (C : 70) - (W : 0)"
print(assert_equals(stock_list(list_of_prod, list_of_cat), expected_result))
################################################################################
list_of_prod = []
list_of_cat = ['B', 'R', 'D', 'X']
expected_result = ""
print(assert_equals(stock_list(list_of_prod, list_of_cat), expected_result))
################################################################################
list_of_prod = ['ROXANNE 102', 'RHODODE 123', 'BKWRKAA 125', 'BTSQZFG 239',␣
 ↪'DRTYMKH 060']
list_of_cat = []
expected_result = ""
print(assert_equals(stock_list(list_of_prod, list_of_cat), expected_result))
################################################################################
list_of_prod = ['ROXANNE 102', 'RHODODE 123', 'BKWRKAA 125', 'BTSQZFG 239',␣
 ↪'DRTYMKH 060']
list_of_cat = ['U', 'V', 'R']
expected_result = "(U : 0) - (V : 0) - (R : 225)"
print(assert_equals(stock_list(list_of_prod, list_of_cat), expected_result))
```

```
True
True
True
True
True
True
```

```
True
```