6.75/10

Group_06_Exercice_04

January 11, 2021

1 Exercice Lab Session 10.12.2020

- ternary operation
  - blueprint
- list comprehension
  - blueprint:
- Exercices

2 1. Ternary Operation:

2.1 1.1. Blueprint:

Ternary operators are more commonly known as conditional expressions in Python. These operators evaluate something based on a condition being true or not.

Here is a blueprint and an example of using these conditional expressions.

```
value_if_true if condition else value_if_false
```

It allows to quickly test a condition instead of a multiline if statement. Often times it can be immensely helpful and can make your code compact but still maintainable.

Example

```
is_nice = True
state = "nice" if is_nice else "not nice"
```

2.2 1.2. Example Ternary Operation

Let's write an algorithm that asks for the user's age. If the user is minor (under the age of 18) then print 'you are not allowed here', otherwise print 'welcome'

[10]:
```
# our code here
age = int(input("Welcome to CinemaXXX, How old are you?"))

if age >= 18:
    print("Welcome!")
else:
    print("You are not allowed here!")
```

```
Welcome to CinemaXXX, How old are you? 17

You are not allowed here!
```

2.3 1.3. Ternary exercice

Place problem: As usual there are problem of not enough space left in the fridge. We want to write a simple program telling if all food will pass into the fridge.

Task Overview: We have to write a function that accepts three parameters:

cap is the amount items the fridge can hold. on is the number of items already in the fridge. wait is the number of items waiting to get into the fridge. If there is enough space, return 0, and if there isn't, return the number items we can't take.

enough(10, 5, 5) 0 # He can fit all 5 items enough(100, 60, 50) 10 # He can't fit 10 out of 50 waiting

[11]:
```
# do it first with a normal if (and not within a function)
cap = 10
on = 5
wait = 5
on_and_wait = on + wait
if on_and_wait <= cap:
    print(0)
else :
    print(on_and_wait - cap)
```

```
0
```

[12]:
```
cap = 100
on = 60
wait = 50

on_and_wait = on + wait
blabla = 0 if on_and_wait <= cap else on_and_wait - cap
print(blabla)
```

```
10
```

[16]:
```
# an now pack it into a function called enough(cap, on, wait) that return the␣
 ↪expected values
def enough(cap, on, wait):
    on_and_wait = on + wait
    return 0 if on_and_wait <= cap else on_and_wait - cap
```

[17]:
```
enough( cap,on,wait )
```

[17]:
```
10
```

3 2. List comprehension

3.1 2.1. Blueprint

List comprehensions provide a short and concise way to create lists. It consists of square brackets containing an expression followed by a for clause, then zero or more for or if clauses. The expressions can be anything, meaning you can put in all kinds of objects in lists. The result would be a new list made after the evaluation of the expression in context of the if and for clauses.

```
variable = [out_exp for out_exp in input_list if out_exp == 2]
```

Note: List comprehension works logically with list. But you can also use comprehension with set, dictionary etc. here for more

3.2 2.2. Example list comprehension:

Let's write an algorithm that save all the passwords that contains the letter 't' in another list

psswd = ['harley','batman','andrew','tigger','sunshine','iloveyou','2000','charlie','robert','thomas','hockey','ranger',

[18]:
```
# again, write is as a normal algorithn with a normal for loop (not within a␣
↪function)
passwords =␣
↪['harley','batman','andrew','tigger','sunshine','iloveyou','2000','charlie','robert','thoma

res = []
for password in passwords :
    if 't' in password:
        res.append(password)
print(res)
```

```
['batman', 'tigger', 'robert', 'thomas', 'starwars', 'klaster']
```

[19]:
```
# write it again but this time in form of a list comprehension
#res_comprehension_u = [password.upper() for password in passwords]
#print(res_comprehension_u)
res_comprehension = [password.upper() for password in passwords if 't' in␣
↪password]
print(res_comprehension)
```

```
['BATMAN', 'TIGGER', 'ROBERT', 'THOMAS', 'STARWARS', 'KLASTER']
```

[23]:
```
# let's pack it in a function called check_t(passwords) that return those␣
↪passwords with 't' in it.
def check_t(passwords):
    return [password.upper() for password in passwords if 't' in password]
```

3.3 2.3. List comprehension exercice

1.1 Write an algorithm that save all even numbers from list into another list with for loop and then with a list comprehension. Put your algorithm into a function called get_even(list_nb) when you are done and return the result list.

```
[24]: # your code here
      # again, write is as a normal algorithn with a normal for loop (not within a␣
       ↪function)

      nb_lst_1 = [2, 38, 49, 37, 43, 40, 49, 59, 44, 25, 36]
      nb_lst_2 = [29, 33, 32, 21, 48, 87, 96, 99, 22, 11, 13]
      res =[]
      for nb in nb_lst_1:
          if nb %2 == 0:
              res.append(nb)
      print(res)
```

```
[2, 38, 40, 44, 36]
```

```
[25]: # write it again but this time in form of a list comprehension
      res_comprehension = [nb for nb in nb_lst_1 if nb %2==0]
      print(res_comprehension)
```

```
[2, 38, 40, 44, 36]
```

```
[27]: # let's pack it in a function get_even(list_nb) that return those even numbers
      def get_even(list_nb):
          return [nb for nb in list_nb if nb %2==0]
```

```
[28]: get_even(nb_lst_2)
```

```
[28]: [32, 48, 96, 22]
```

4 3. Exercices

4.1 Exercice 1

We want returns the *index* of the numerical element that lies between the other two elements.

The input to the function will always be an array of three distinct numbers.

For example:

gimme([2, 3, 1]) => 0

2 is the number that fits between 1 and 3 and the index of 2 in the input array is 0.

Another example (just to make sure it is clear):

gimme([5, 10, 14]) => 1

10 is the number that fits between 5 and 14 and the index of 10 in the input array is 1.

here are some functions that you might use here

```
min() # see above
max() # see above

my_list.index(value) # index()  will give you the index of 'value' if value is in the array. I
```

[29]:
```
my_list = [1,10,5]
maximum = max(my_list)
minimum = min(my_list)
#my_list.index(14)
index_maximum = my_list.index(maximum)
index_minimum = my_list.index(minimum)

possibility =[0,1,2]
exists = [index_minimum, index_maximum]

res = 0

for pos in possibility:
    if pos not in exists:
        res = pos
print(res)
```

```
2
```

[30]:
```
# write it again but this time in form of a list comprehension
res_comprehension = [pos for pos in possibility if pos not in exists]
print(res_comprehension)
```

```
[2]
```

[123]:
```
# put it into a function
def index_list(x):
    return [a.index(i) for i in a if i < Mx_a and i > Mi_a]
print(a.index(i))
```

```
2
```

4.2 Exercice 2

0.75/1.5 pt

Write a function that return the sum of the even values of a sequence.

Examples:

[4, 3, 1, 2, 5, 10, 6, 7, 9, 8] –> 30 # because 4 + 2 + 10 + 6 + 8 = 30

[] –> 0

[15]:
```
# your code here
# [2,3,5,6,9,10,8] # should be  26
```

```
nb_1 = [4, 3, 1, 2, 5, 10, 6, 7, 9, 8]
new_nb = []
for number in nb_1:
    if number % 2 == 0:
        new_nb.append (number)
        sum_even_nb = sum (new_nb)
print (sum_even_nb)
```

```
30
```

[16]:
```
#sum_even_nb = sum (new_nb)
#new_nb.append(number)
sum_even = [number for number in nb_1 if number % 2 == 0]
sum_even_nb = sum(new_nb)
print (sum_even_nb)
```

```
30
```

[17]:
```
nb_2 = [3,33,6,7,39,210,18]
def sum_even_all(x):
    #new_nb.append (number)
    new_nb = []
    #sum_even = [number for number in nb_2 if number % 2 == 0 ]
    sum_even_nb = sum(new_nb)
    return (sum_even_nb)
sum_even_all(nb_2)
```

[17]:
```
0
```

4.3 Exercice 3

1/1.5 pt

IT-Bar recommends you drink 1 glass of water per standard drink so you're not hungover tomorrow morning.

Your fellow coders have bought you several drinks tonight in the form of a string. Return a string suggesting how many glasses of water you should drink to not be hungover.

Examples

"1 beer" –> "1 glass of water" because you drank one standard drink

"1 shot, 5 beers, 2 shots, 1 glass of wine, 1 beer" –> "10 glasses of water" because you drank ten standard drinks

Note:

To keep the things simple, we'll consider that any "numbered thing" in the string is a drink. Even "1 bear" -> "1 glass of water"; or "1 chainsaw and 2 pools" -> "3 glasses of water"...

here are some functions that you might use here

```
var.isdigit() # return True it the variable 'var' is a digig, False if 'var' is not a digit
```

```
var1 = 5
var1.isdigit() # will return True because 5 is a digit

var2 = 'n'
var2.isdigit() # will return False because 'n' is not a digit

var3 = 'test'
var3.isdigit() # will return False because 'test' is not a digit
```

[1]:
```
drink = "6 beer horns and 1 shots of rum, 5 shots of rum, 4 shots of rum, 1␣
 ↪beers, 5 cups of coffee, 1 beer horns, 2 shots"
sum_d = []
for i in range(len(drink)):
    if (drink[i].isdigit()) == True:
        sum_d.append(drink[i])
sum = 0
for j in range(len(sum_d)):
    if sum_d[j] >= '0' and sum_d[j] <= '9':
        sum = sum + int(sum_d[j])
print ('{} glasses of water.'.format(sum))
```

```
25 glasses of water.
```

[2]:
```
#sum_d = {}
for i in range(len(drink)):
    if (drink[i].isdigit()) == True:            # 0 and int(l_drink[i]) <= 9:
        sum_d = (''.insert(drink[i]))
print(sum_d)
```

```
<generator object <genexpr> at 0x000002478C492C80>
```

[ ]:
```
drink = [1, 5, 2, 1, 1]
b = sum(drink)
b
```

4.4 Exercice 4

2/2 pts

From an Integer list, create a new list by adding each consecutive pair of the list.

Examples:

[1, 1, 1, 1] –> [2, 2, 2] # [1+1, 1+1, 1+1]

[1, 2, 3, 4] –> [3, 5, 7] # [1+2, 2+3, 3+4]

[1, 10, 100] –> [11, 110] # [1+10, 10+100]

[3]:
```
# your code here
list_1 = [1,2,3,4,5,6,7,8]
list_2 = []
for i in range(len(list_1)-1):
    #for l in list_1:
    l = list_1[i] + list_1[i+1]
    list_2.append(l)

print((list_2))
```

```
[3, 5, 7, 9, 11, 13, 15]
```

[4]:
```
#list_2.append(l)

list_2 = [list_1[i] + list_1[i+1] for i in range(len(list_1)-1) ]
#l = list_1[i] + list_1[i+1]
#list_2.append(l)
print(list_2)
```

```
[3, 5, 7, 9, 11, 13, 15]
```

[5]:
```
make_new_list=[1,2,3,4]
def new_lst(x):
    return [list_1[i] + list_1[i+1] for i in range(len(make_new_list)-1)]
new_lst (make_new_list)
```

[5]:
```
[3, 5, 7]
```

4.5 Exercice 5

1.5/2.5 pts

Given an array of integers , Find the maximum product obtained from multiplying 2 adjacent numbers in the array.

Notes Array/list size is at least 2.

Array/list numbers could be a mixture of positives, negatives also zeroes .

Examples:

adjacentElementsProduct([1, 2, 3]); ==> return 6

Explanation:

The maximum product obtained from multiplying 2 * 3 = 6, and they're adjacent numbers in the array.

adjacentElementsProduct([9, 5, 10, 2, 24, -1, -48]); ==> return 50

Explanation:

Max product obtained from multiplying 5 * 10 = 50.

here are some functions that you might use here

```
max() # return the max from a list
max([2,3,4,5]) # will return 5 because 5 is the max in the list

min() # same as max but it will return the minimum
```

[6]:
```
list_2 = [5, 1, 2, 3, 1, 4,3,6,1,8,2]
res = []
for i in range(len(list_2)-1):
    new_elm_1 = list_2[i] * list_2[i+1]
    res.append(new_elm_1)
    res_nb = max(res)
#print(res)
print(res_nb)
```

```
18
```

[7]:
```
new_elm = list_2[i] * list_2[i+1]
res_nb = max(res)
print_nb = [res_nb for i in range(len(list_2)-1)]
print(res_nb)
```

```
18
```

[8]:
```
def multiply():
    res_nb = max(res)
    return[res_nb for i in range(len(list_2)-1)]

print(res_nb)
```

```
18
```

[ ]:
```
# this is for testing our answer

assert_equals(adjacent_element_product([5, 8]), 40)
assert_equals(adjacent_element_product([1, 2, 3]), 6)
assert_equals(adjacent_element_product([1, 5, 10, 9]), 90)
assert_equals(adjacent_element_product([4, 12, 3, 1, 5]), 48)
assert_equals(adjacent_element_product([5, 1, 2, 3, 1, 4]), 6)

# "Both positive and negative values"
assert_equals(adjacent_element_product([3, 6, -2, -5, 7, 3]), 21)
assert_equals(adjacent_element_product([9, 5, 10, 2, 24, -1, -48]), 50)
assert_equals(adjacent_element_product([5, 6, -4, 2, 3, 2, -23]), 30)
assert_equals(adjacent_element_product([-23, 4, -5, 99, -27, 329, -2, 7,␣
 ↪-921]), -14)
assert_equals(adjacent_element_product([5, 1, 2, 3, 1, 4]), 6)

# "Contains zeroes"
assert_equals(adjacent_element_product([1, 0, 1, 0, 1000]), 0)
assert_equals(adjacent_element_product([1, 2, 3, 0]), 6)
```

4.6 Exercice 6

1.5/2.5 pts

Your task is to write a function called valid_spacing() or validSpacing() which checks if a string has valid spacing. The function should return either True or False.

For this kata, the definition of valid spacing is one space between words, and no leading or trailing spaces. Below are some examples of what the function should return.

[15]:
```
# your code here
sentence = 'hkg '
sentence_split = sentence.split(" ")
if sentence_split[0] == '' or sentence_split[len(sentence_split)-1] == '':
    print(False)
else:
    print(True)
```

```
False
```

[16]:
```
print(False if sentence_split[0] == '' or sentence_split[len(sentence_split)-1]␣
 ↪== '' else True)
sentence_split = sentence.split(" ")
```

```
False
```

[18]:
```
def sentence_bool(x):
    x_split = x.split(" ")
    return[False if x_split[0] == '' or x_split[len(x_split)-1] == '' else True]
sentence_bool('Helloworld ')
```

[18]:
```
[False]
```

[ ]:
```
assert_equals(valid_spacing('Hello world'),True)
assert_equals(valid_spacing(' Hello world'),False)
assert_equals(valid_spacing('Hello  world '),False)
assert_equals(valid_spacing('Hello'),True)
assert_equals(valid_spacing('Helloworld'),True)
assert_equals(valid_spacing(''),True)
assert_equals(valid_spacing(' '),False)
```