#1

Повернемося до завдання про систему оцінок в університеті, які мають такий вигляд:

| **Оцінка** | **Бали** | **Оцінка ECTS** | **Пояснення** |
| --- | --- | --- | --- |
| 1 | 0-34 | F | Unsatisfactorily |
| 2 | 35-59 | FX | Unsatisfactorily |
| 3 | 60-66 | E | Enough |
| 3 | 67-74 | D | Satisfactorily |
| 4 | 75-89 | C | Good |
| 5 | 90-95 | В | Very good |
| 5 | 96-100 | A | Perfectly |

Минулого разу ми реалізували дві функції. Перша - get\_grade, приймає ключ в оцінці ECTS і повертає відповідну п'ятибальну оцінку (перший стовпчик таблиці). Друга - get\_description, теж приймає ключ у оцінки ECTS, але повертає пояснення оцінки у текстовому форматі (останній стовпчик таблиці). На неіснуючий ключ функції повинні повертати значення None.

Реалізуйте функцію вищого порядку get\_student\_grade, яка приймає параметр option. Якщо він дорівнює значенням "grade", то функція повертає функцію get\_grade, а якщо його значення дорівнює "description", то повертає функцію get\_description. Якщо параметр за значенням не співпав із заданими, то функція get\_student\_grade повинна повертати значення None.

*def* get\_grade(*key*):

    grade = {"A": 5, "B": 5, "C": 4, "D": 3, "E": 3, "FX": 2, "F": 1}

    return grade.get(*key*, None)

*def* get\_description(*key*):

    description = {

        "A": "Perfectly",

        "B": "Very good",

        "C": "Good",

        "D": "Satisfactorily",

        "E": "Enough",

        "FX": "Unsatisfactorily",

        "F": "Unsatisfactorily",

    }

    return description.get(*key*, None)

*def* get\_student\_grade(*option*):

    if *option* == "grade":

        return get\_grade

    elif *option* == "description":

        return get\_description

    else:

        return None

print(get\_student\_grade("grade"))

#2

**ОБЛАСТІ ВИДИМОСТІ (LEGB)**

 (LOCAL, ENCLOSED, GLOBAL, BUILT INS)

Реалізуйте функцію get\_discount\_price\_customer для розрахунку ціни на товар інтернет-магазину з урахуванням знижки клієнта.

Функція приймає два параметри:

* price — ціна продукту
* customer — словник з даними клієнта такого виду: {"name": "Dima"} або {"name": "Boris", "discount": 0.15}

Ви маєте глобальну змінну DEFAULT\_DISCOUNT, яка визначає знижку для клієнта, якщо у нього немає поля discount.

Функція get\_discount\_price\_customer має повертати нову ціну товару для клієнта.

Нагадаємо, що дисконт discount - це дробове число від 0 до 1. І ми під знижкою розуміємо коефіцієнт, який визначає величину ціни. І на цю величину ми знижуємо підсумкову ціну товару: price = price \* (1 - discount).

DEFAULT\_DISCOUNT = 0.05

#customer = {"name": "Dima"}

customer = {"name": "Boris", "discount": 0.15}

*def* get\_discount\_price\_customer(*price*, *customer*):

    discount = *customer*.get("discount", DEFAULT\_DISCOUNT)

*price* = *price* \* (1 - discount)

    return *price*

print(get\_discount\_price\_customer(100, customer))

#3

## ЗАМИКАННЯ

Створіть функцію caching\_fibonacci(), яка матиме кеш із попередньо обчисленими значеннями чисел Фібоначі. Усередині вона містить функцію fibonacci(n), яка безпосередньо і обчислюватиме саме число Фібоначчі. Функція caching\_fibonacci() повертає функцію fibonacci

Якщо число Фібоначчі зберігається у словнику cache, то функція fibonacci повертає число з кеша. Якщо його немає у кеші, то ми обчислюємо число і поміщаємо його в кеш, і повертаємо з функції fibonacci.

*def* caching\_fibonacci():

    cache = {0: 0, 1: 1}

*def* fibonacci(*n*):

        if *n* in cache:

            return cache.get(*n*)

        else:

            res = fibonacci(*n* - 1) + fibonacci(*n* - 2)

            new\_item = {*n*: res}

            cache.update(new\_item)

            print(cache)

            return res

    return fibonacci

smthing = caching\_fibonacci()

result = smthing(6)

print(result)

result = smthing(7)

print(result)

#4

## КАРРУВАННЯ

Повернемося до завдання розрахунку ціни з урахуванням дисконту та розберемо підхід із позиції карування. Створіть функцію discount\_price(discount), яка визначатиме в собі та повертатиме функцію розрахунку реальної ціни з урахуванням знижки.

Виклик функції discount\_price(discount) поверне функцію, яка розраховує ціну на товар зі знижкою, що дорівнює discount .

Наприклад:

cost\_15 = discount\_price(0.15)

cost\_10 = discount\_price(0.10)

cost\_05 = discount\_price(0.05)

price = 100

print(cost\_15(price))

print(cost\_10(price))

print(cost\_05(price))

Повинен вивести:

85.0

90.0

95.0

*def* discount\_price(*discount*):

*def* price(*price*):

        return *price* \* (1 - *discount*)

    return price

cost\_15 = discount\_price(0.15)

cost\_10 = discount\_price(0.10)

cost\_05 = discount\_price(0.05)

price = 100

print(cost\_15(price))

print(cost\_10(price))

print(cost\_05(price))

#5

## ДЕКОРАТОРИ

У модулі 5 ми написали функцію sanitize\_phone\_number для нормалізації рядка з телефонним номером. Нагадаємо, що при отриманні рядків

" +38(050)123-32-34",

" 0503451234",

"(050)8889900",

"38050-111-22-22",

"38050 111 22 11 ",

Ми отримували наступний вивід:

380501233234

0503451234

0508889900

380501112222

380501112211

Уявіть, що в іншому місці програми у нас виникла вимога зробити висновок у форматі

+380501233234

+380503451234

+380508889900

+380501112222

+380501112211

І тут ідеально підійде створення декоратора для функції sanitize\_phone\_number. Декоратор повинен додавати для коротких номерів префікс +38, а для повного міжнародного номера (з 12 символом) - тільки знак +. Реалізуйте декоратор format\_phone\_number для функції sanitize\_phone\_number з необхідним функціоналом.

*def* format\_phone\_number(*func*):

*def* inner(*phone*):

        income = *func*(*phone*)

        if len(income) == 12:

            outcome = "+" + income

        elif len(income) == 10:

            outcome = "+38" + income

        return outcome

    return inner

@format\_phone\_number

*def* sanitize\_phone\_number(*phone*):

    new\_phone = (

*phone*.strip()

        .removeprefix("+")

        .replace("(", "")

        .replace(")", "")

        .replace("-", "")

        .replace(" ", "")

    )

    return new\_phone

print(sanitize\_phone\_number("    +(050)123-32-34"))

#6

## ІТЕРАТОРИ/ГЕНЕРАТОРИ (КЛЮЧОВЕ СЛОВО YIELD)

Нехай є рядок з числами (з метою спрощення числа лише цілі), що визначає якісь частини загального доходу. Наприклад,

"The resulting profit was: from the southern possessions $ 100, from the northern colonies $500, and the king gave $1000."

Необхідно реалізувати функцію generator\_numbers, яка буде парсити рядок і знаходити всі цілі числа в ньому та працювати як генератор, який буде віддавати зазначені числа при зверненні до нього у циклі.

З парсингом рядків ми вже зіштовхувалися виконуючи завдання модуля 7, коли розбивали на лексеми арифметичний вираз

Функція generator\_numbers(string="") безпосередньо розпарсює рядок і за допомогою yield повертає поточне число.

Функція sum\_profit(string) підсумовує числа, отримані від generator\_numbers, та повертає загальну суму прибутку з рядка.

*def* generator\_numbers(*string*=""):

    l = len(*string*)

    i = 0

    while i < l:

        strings\_int = ''

        a = *string*[i]

        while '0' <= a <= '9':

            strings\_int += a

            i += 1

            if i < l:

                a = *string*[i]

            else:

                break

        if strings\_int != '':

            yield int(strings\_int)

        i += 1

*def* sum\_profit(*string*):

    nums = generator\_numbers(*string*)

    sum = 0

    for b in nums:

        sum += b

    print(sum)

    return sum

#7

## ЛЯМБДА-ФУНКЦІЇ ТА MAP

Є список name з іменами користувачів, але всі починаються з малої літери.

name = ["dan", "jane", "steve", "mike"]

Розробіть функцію normal\_name, яка приймає список імен та повертає теж список імен, але вже з правильними іменами з великої літери.

['Dan', 'Jane', 'Steve', 'Mike']

Необхідно використовувати функцію map. Не забудьте, що необхідно виконати перетворення типів для map.

*def* normal\_name(*list\_name*):

    new\_list = []

    for i in map(*lambda* *x*: *x*, *list\_name*):

        ch = ord(i[0]) - 32

        new\_list.append((chr(ch))+i[1:])

    return new\_list

list\_name = ["dan", "jane", "steve", "mike"]

normal\_name(list\_name)

#8

## ОТРИМАТИ EMAIL КОРИСТУВАЧІВ

Є список contacts, елементи якого - словники контактів наступного виду:

{

"name": "Allen Raymond",

"email": "nulla.ante@vestibul.co.uk",

"phone": "(992) 914-3792",

"favorite": False,

}

Словник містить ім'я користувача, його email, телефонний номер та властивість - обраний контакт чи ні.

Розробіть функцію get\_emails, яка отримує у параметрі список list\_contacts та повертає список, який містить електронні адреси всіх контактів зі списку list\_contacts. Використовуйте функцію map.

*def* get\_emails(*list\_contacts*) -> list:

    emails\_list = []

    for i in map(*lambda* *item*: *item*["email"], *list\_contacts*):

        emails\_list.append(i)

    return list(emails\_list)

еще вариант:

def get\_emails(list\_contacts):

return list(map(lambda x: x.get('email'), list\_contacts))

№9

## ФУНКЦІЯ FILTER

На початку четвертого модуля ми вирішували завдання виплат за комунальними платежами. Вони являли собою список payment з додатними та від'ємними значеннями. Створіть функцію positive\_values та за допомогою функції filter відфільтруйте список payment за додатними значеннями, та поверніть його з функції.

payment = [100, -3, 400, 35, -100]

*def* positive\_values(*list\_payment*):

    new\_list = []

    for i in filter(*lambda* *x*: *x* > 0, *list\_payment*):

        new\_list.append(i)

    return new\_list

list\_payment = [100, -3, 400, 35, -100]

print(positive\_values(list\_payment))

#10

## ЗАВДАННЯ: ВІДФІЛЬТРУВАТИ КОРИСТУВАЧІВ

Є список contacts, елементи якого - словники контактів наступного виду:

{

"name": "Allen Raymond",

"email": "nulla.ante@vestibul.co.uk",

"phone": "(992) 914-3792",

"favorite": False,

}

Словник містить ім'я користувача, його email, телефонний номер та властивість - обраний контакт чи ні.

Створіть функцію get\_favorites(contacts), яка повертатиме список, який містить лише обрані контакти. Використовуйте при цьому функцію filter, щоб відфільтрувати по полю favorite лише обрані контакти.

*def* get\_favorites(*contacts*):

    return list(filter(*lambda* *x*: *x*.get('favorite') == True, *contacts*))

contacts = [

    {"name": "Allen Raymond",

     "email": "nulla.ante@vestibul.co.uk",

     "phone": "(992) 914-3792",

     "favorite": False, },

    {"name": "Alyona Raymond",

     "email": "yona@vestibul.com.uk",

     "phone": "(992) 914-3792",

     "favorite": False, },

    {"name": "Alla Raymond",

     "email": "alla\_1@vestibul.com",

     "phone": "(992) 914-3792",

     "favorite": True, },

    {"name": "Oleg Raymond",

     "email": "111\_oleg@oleg.com.on",

     "phone": "(992) 914-3792",

     "favorite": False, },

    {"name": "Olga Raymond",

     "email": "Olga.125@google.org",

     "phone": "(992) 914-3792",

     "favorite": True, },

]

print(get\_favorites(contacts))

#11

## ФУНКЦІЯ REDUCE

Для списку numbers підрахувати суму елементів за допомогою функції reduce.

numbers = [3, 4, 6, 9, 34, 12]

Створіть функцію sum\_numbers(numbers), результатом виконання якої буде сума чисел всіх елементів списку numbers.

from functools import reduce

*def* sum\_numbers(*numbers*):

    result = reduce((*lambda* *x*, *y*: *x* + *y*), *numbers*)

    return result

numbers = [1, 2, 3, 4]

print(sum\_numbers(numbers))

#12

## ЗАВДАННЯ: ЗАСТОСУВАННЯ REDUCE

Повернемося до нашого першого завдання з четвертого модуля і перепишемо його за допомогою функції reduce.

payment = [1, -3, 4]

def amount\_payment(payment):

sum = 0

for value in payment:

if value > 0:

sum = sum + value

return sum

Нагадаємо умову. У нас є список показань заборгованостей з комунальних послуг наприкінці місяця. Заборгованості можуть бути від'ємними — у нас переплата, чи додатними, якщо необхідно сплатити за рахунками. За допомогою reduce підсумуйте додатні значення та поверніть з функції amount\_payment суму платежу в кінці місяця.

from functools import reduce

*def* amount\_payment(*payment*):

    new\_list = []

    for i in filter(*lambda* *x*: *x* > 0, *payment*):

        new\_list.append(i)

    result = reduce((*lambda* *x*, *y*: *x* + *y*), new\_list)

    return result

payment = [1, -3, 4]

print(amount\_payment(payment))