Dekoratory oraz programowanie funkcyjne

1. Dekoratory funkcji

(**1 pkt**) Proszę zdefiniować dekorator o nazwie @argumenty(arg_1,arg_2, ...,arg_N) za pomocą którego będzie można określać wartości domyślne argumentów funkcji. Pierwszy z arg_i, który nie został przyporządkowany do argumentów funkcji jest traktowany jako wartość, którą funkcja zwraca gdy w ciele funkcji nie określono co ma zwracać. Zakładamy, że zawartość listy argumentów dekoratora można zmieniać, jak to pokazano w w linii 24 lub 28. **Skryptowi mają towarzyszyć testy jednostkowe**.

```
class Operacje:
  argumentySuma=[4,5]
  argumentyRoznica=[4,5,6]

  @argumenty(argumentySuma)
  def suma(a,b,c):
    print "%d+%d+%d=%d" % (a,b,c,a+b+c)

  @argumenty(argumentyRoznica)
  def roznica(x,y):
    print "%d-%d=%d" % (x,y,x-y)

op=Operacje()
op.suma(1,2,3) #Wypisze: 1+2+3=6
op.suma(1,2) #Wypisze: 1+2+4=7 - 4 jest pobierana z tablicy 'argumentySuma'
op.suma(1) #Wypisze: 1+4+5=10 - 4 i 5 są pobierane z tablicy 'argumentySuma'
op.suma() #TypeError: suma() takes exactly 3 arguments (2 given)
op.roznica(2,1) #Wypisze: 2-1=1
op.roznica(2) #Wypisze: 2-4=-2
wynik=op.roznica() #Wypisze: 4-5=-1
print wynik #Wypisze: 6

#Zmiana zawartości listy argumentów dekoratora  dla metody 'suma'
op['suma']=[1,2]
#oznacza, że   argumentySuma=[1,2]

#Zmiana zawartości listy argumentów dekoratora  dla metody 'roznica'
op['roznica']=[1,2,3]
#oznacza, że   argumentyRoznica=[1,2,3]
```

2. Programowanie funkcyjne

Wszystkie skrypty należy zaimplementować używając, **tylko i wyłącznie**, **podejścia funkcyjnego** — opis 1, opis 2 oraz opis 3

1. Przeanalizuj, poniższe, algorytmy znajdowania sumy pierwszych 10 liczb naturalnych, których kwadrat jest podzielny przez 5

Podejście imperatywne

```
n = 1
num_elements = 0
suma = 0
while num_elements < 10:
    if (n**2 % 5) == 0:
        suma += n
        num_elements += 1
    n += 1

print(suma)
```

Podejście funkcyjne

```
import itertools, functools, sys
dane = filter(lambda x: x**2 % 5 == 0, range(1, sys.maxsize))
dane = itertools.islice(dane, 10)
suma = functools.reduce(lambda x, y: x+y, dane)
print(suma)

# To samo co powyżej, ale zapisane inaczej
suma = functools.reduce(
        lambda x, y: x+y, itertools.islice(
            filter(
                lambda x: x**2 % 5 == 0, range(1, sys.maxsize)
            ), 10
        )
    )
print(suma)
```

2. (**1 pkt**) Stwórz jednolinijkowiec, który dla wielowierszowego tekstu wczytanego z klawiatury i zakończonego `Ctrl+D` (Linux, macOS) / `Ctrl+z` (MS Windows), oblicza ilość wystąpień długości napisów. Przykład działania skryptu:

```
$ python3 -c 'Treść jednolinijkowca'
qq ffaf
1 1
^D
{1:2, 2:1, 4:1}
```

Składnia `{ długośćNapisu`$_1$`:krotnośćWystąpienia1, długośćNapisu`$_2$`:krotnośćWystąpienia2, ... }`

3. (**1 pkt**) Stwórz skrypt, który oblicza sumaryczną ilość wystąpień liczb parzystych zawartych w plikach tekstowych, których nazwy są wymienione w linii komend. Należy przyjąć, że poszczególne liczby są rozdzielone białymi znakami.
Przykład:

plik1.txt

```
44 6 7
```

plik2.txt

```
123 45
```

```
89 7
```

```
102
763  9
```

Po wydaniu komendy `skrypt.py plik1.txt plik2.txt` program powinien wypisać: 3

4. (**2 pkt**) Korzystając z podejścia funkcyjnego stwórz skrypt o funkcjonalności, która zostanie podana na początku ćwiczeń