Laboratorium **Programowanie w języku Python 2** Wydział Elektrotechniki Automatyki I Informatyki Politechnika Świętokrzyska	
Studia: **Stacionarne I stopnia**	Kierunek: **Informatyka**
Data wykonania: **25.03.2021**	Grupa: **3ID16B**
Imię I nazwisko: **Arkadiusz Więcław**	Temat ćwiczenia: **Wyrażenie generatorowe** **Dodatkowe iteratory** **Deskryptory**

Zad 1:

Przykład 1

```
unique_characters = {'E', 'D', 'M', 'O', 'N', 'S', 'R', 'Y'}
gen = (ord(c) for c in unique_characters)
gen
next(gen)
next(gen)
print("\nPrzykład 1 = " ,tuple(ord(c) for c in unique_characters))
```

Wynik:

```
Przykład 1 =  (77, 83, 89, 68, 69, 78, 79, 82)
```

Przykład 2

```
import itertools
perms = itertools.permutations([1, 2, 3], 2)
print("\nPrzykład 2 = ")
print(next(perms))
print(next(perms))
print(next(perms))
print(next(perms))
print(next(perms))
print(next(perms))
```

Wynik:

```
Przykład 2 =

(1, 2)

(1, 3)

(2, 1)

(2, 3)

(3, 1)

(3, 2)
```

Przykład 3

```
import itertools
perms = itertools.permutations('ABC', 3)
print("\nPrzykład 3 = ")
print(next(perms))
print(next(perms))
print(next(perms))
print(next(perms))
print(next(perms))
print(next(perms))
#print(next(perms))
print(list(itertools.permutations('ABC', 3)))
```

Wynik:

```
Przykład 3 =

('A', 'B', 'C')

('A', 'C', 'B')

('B', 'A', 'C')

('B', 'C', 'A')

('C', 'A', 'B')

('C', 'B', 'A')

[('A', 'B', 'C'), ('A', 'C', 'B'), ('B', 'A', 'C'), ('B', 'C', 'A'),
('C', 'A', 'B'), ('C', 'B', 'A')]
```

Przykład 4

```
import itertools
print("\nPrzykład 4 = ")
print("\nWariant 1 = ",list(itertools.product('ABC', '123')))
print("\nWariant 2 = ",list(itertools.combinations('ABC', 2)))
```

Wynik:

```
Przykład 4 =

Wariant 1 =  [('A', '1'), ('A', '2'), ('A', '3'), ('B', '1'), ('B',
'2'), ('B', '3'), ('C', '1'), ('C', '2'), ('C', '3')]

Wariant 2 =  [('A', 'B'), ('A', 'C'), ('B', 'C')]
```

Przykład 5

```
names=('Dora ','Ethan ','Wesley ','John ','Anne ','Mike', 'Chris
','Sarah ','Alex ','Lizzie ')
```

```
print("\nPrzykład 5 = ")
print(names)
names = [name.rstrip() for name in names]
print(names)
names = sorted(names)
print(names)
names = sorted(names, key=len)
print(names)
import itertools
groups = itertools.groupby(names, len)
print(names)
list(groups)
groups = itertools.groupby(names, len)
for name_length, name_iter in groups:
    print('Names with {0:d} letters:'.format(name_length))
    for name in name_iter:
        print(name)
```

Wynik:

```
Przykład 5 =

('Dora ', 'Ethan ', 'Wesley ', 'John ', 'Anne ', 'Mike', 'Chris ',
'Sarah ', 'Alex ', 'Lizzie ')

['Dora', 'Ethan', 'Wesley', 'John', 'Anne', 'Mike', 'Chris', 'Sarah',
'Alex', 'Lizzie']

['Alex', 'Anne', 'Chris', 'Dora', 'Ethan', 'John', 'Lizzie', 'Mike',
'Sarah', 'Wesley']

['Alex', 'Anne', 'Dora', 'John', 'Mike', 'Chris', 'Ethan', 'Sarah',
'Lizzie', 'Wesley']

['Alex', 'Anne', 'Dora', 'John', 'Mike', 'Chris', 'Ethan', 'Sarah',
'Lizzie', 'Wesley']

Names with 4 letters:

Alex

Anne

Dora

John

Mike
```

```
Names with 5 letters:

Chris

Ethan

Sarah

Names with 6 letters:

Lizzie

Wesley
```

Przykład 6

```
print("\nPrzykład 6 = ")
class MyDescriptor(object):
    """A data descriptor that sets and returns values
    normally and prints a message logging their access.
    """
def __init__(self, initval=None, name='var'):
    self.val = initval
    self.name = name
def __get__(self, obj, objtype):
    print('Retrieving', self.name)
    try:
        return self.val
    except:
        print('Var x is not declared')
def __set__(self, obj, val):
    print('Updating', self.name)
    self.val = val
def __delete__(self, obj):
    print('Deleting', self.name)
    try:
        del self.val
    except:
        print('Var x is not declared')
class MyClass(object):
    x = MyDescriptor()
    y = 5
m = MyClass()
print(m.x)
m.x = 20
print(m.x)
print(m.y)
```

```
del m.x
print(m.x)
m.x=4
print(m.x)
```

Wynik:

```
Przykład 6 =

<__main__.MyDescriptor object at 0x0000028CF8870A60>

20

5

<__main__.MyDescriptor object at 0x0000028CF8870A60>

4
```

Przykład 7

```
class Descriptor(object):
    def __init__(self, name =''):
        self.name = name
    def __get__(self, obj, objtype):
        return "{}for{}".format(self.name, self.name)
    def __set__(self, obj, name):
        if isinstance(name, str):
            self.name = name
        else:
            raise TypeError("Name should be string")
class GFG(object):
    name = Descriptor()
g = GFG()
g.name = "Computer"
print("\nPrzykład 7 = ", g.name)
```

Wynik:

```
Przykład 7 =  ComputerforComputer

40
```

Przykład 8

```
class Descriptor:
    def __init__(self):
        self.__fuel_cap = 0
    def __get__(self, instance, owner):
        return self.__fuel_cap
    def __set__(self, instance, value):
        if isinstance(value, int):
```

```
            print(value)
        else:
            raise TypeError("Fuel Capacity can only be an integer")
        if value < 0:
            raise ValueError("Fuel Capacity can never be less than
zero")
            self.__fuel_cap = value
    def __delete__(self, instance):
        del self.__fuel_cap
class Car:
    fuel_cap = Descriptor()
    def __init__(self,make,model,fuel_cap):
        self.make = make
        self.model = model
        self.fuel_cap = fuel_cap
    def __str__(self):
        return "{0} model {1} with a fuel capacity of {2}
ltr.".format(self.make,self.model,self.fuel_cap)
car2 = Car("BMW","X7",40)
print("\nPrzykład 8 = ", car2)
```

Wynik:

```
Przykład 8 =  BMW model X7 with a fuel capacity of 0 ltr.
```

Zad 2:

Przykład 1

```
#Przyklad za pomoca wyrazenia generatorowego wypisuje dlugosc kazdego
elemnetu tupli.
string_characters = ("Mateusz", "Sebastian", "Arkadiusz", "Oscar",
"Artor", "Damian")
gen = (len(w) for w in string_characters )
print("\nPrzykład 1 = ", tuple(gen))
```

Wynik:

```
Przykład 1 =  (7, 9, 9, 5, 5, 6)
```

Przykład 2

```
#Przyklad za pomoca wyrazenia generatorowego zamienia liczby zawarte w
liscie na znaki.
liczby = [789, 331, 200, 960, 78, 231]
print("\nPrzykład 2 = ", list(chr(l) for l in liczby))
```

Wynik:

```
Przykład 2 =  ['ʹ', 'ŋ', 'È', 'π', 'N', 'ç']
```

Zad 3:

Przykład 1

```
#Przyklad wykorzystuje iterator count  wyswietla 16 liczb po kolej
dodajac do nastepnej 4.
from itertools import count

print("\nPrzykład 1 = ")
count_itra = count(step=4)
for i in range(16):
        print(next(count_itra))
```

Wynik:

```
Przykład 1 =

0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60
```

Przykład 2

```
#Przyklad wykorzystuje iterator chain ktory wyswietla zawartosc trzech
tablic
from itertools import chain

print("\nPrzykład 2 = ")
tab1 = ["fghj", "bbnjm", "hjhj"]
tab2 = ["serer", "gownoo", "mars"]
tab3 = ["tylko","dfer","sasks"]
for x in chain(tab1, tab2, tab3):
    print(x)
```

Wynik:

```
Przykład 2 =

fghj bbnjm hjhj serer gownoo mars tylko dfer sasks
```

Przykład 3

```
#Przyklad wykorzystuje iterator starmap ktory wybiera w tym przypadku
najwieksza wartosc
#z trzech list znajdujacych sie wewnatrz jednej listy
from itertools import starmap

liczby = [ [5, 7 ,4], [6,8,3], [75,35,90] ]
wynik = starmap(max, liczby)
print("\nPrzykład 3 = ", list(wynik))
```

Wynik:

```
Przykład 3 =  [7, 8, 90]
```

Zad 4:

Przykład 1

```
#Przyklad wykorzystuje iterator chunked ktory w tym przypadku  dzieli
liste na dwie podlisty
from more_itertools import chunked

tab = ["fghj", "bbnjm", "hjhj", "serer", "gownoo", "mars"]
print("\nPrzykład 1 = ", list(chunked(tab, 3)))
```

Wynik:

```
Przykład 1 =  [['fghj', 'bbnjm', 'hjhj'], ['serer', 'gownoo', 'mars']]
```

Przykład 2

```
#Przyklad wykorzystuje iterator distribute ktory  w tym przypadku
dzieli liste na dwie listy 3 elementowe
from more_itertools import distribute

t1, t2 = distribute(2, ["fghj", "bbnjm", "hjhj", "serer", "gownoo",
"mars"])
print("\nPrzykład 2 = ")
print(list(t1))
print(list(t2))
```

Wynik:

```
Przykład 2 =

['fghj', 'hjhj', 'gownoo']

['bbnjm', 'serer', 'mars']
```