PRZYKŁADY DZIAŁANIA LISTA 7

Zadanie 1

a. acronym

Funkcja przyjmuje na wejściu listę ciągów znaków zwraca akronim

zbudowany z tych ciągów znaków

```
4   print(list(acronym(['Zakład', 'Ubezpieczeń', 'Społecznych'])))
5   #print(list(acronym(7)))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
['Z', 'U', 'S']
```

W przypadku podania pustej listy jako parametr, wynikiem jest również pusta lista

```
#print(list(acronym(['Zakład', 'Ubezpieczeń', 'Społecznych'])))
print(list(acronym([])))
#print(list(acronym(7)))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
[]
```

W przypadku podania jako parametr innej wartości niż lista dostajemy komunikat „ Nie podano listy znaków jako parametr”.

```
4   #print(list(acronym(['Zakład', 'Ubezpieczeń', 'Społecznych'])))
5   #print(list(acronym([])))
6   print(list(acronym(7)))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
Nie podano listy znaków jako parametr
Traceback (most recent call last):
  File "/home/student/Pulpit/Lab7/main.py", line 6, in <module>
    print(list(acronym(7)))
TypeError: 'NoneType' object is not iterable
```

b. median

Liczy medianę i w przypadku nieparzystej jak i parzystej liczby elementów

```
8   print(median([1,1,19,2,3,4,4,5,1]))
9   print(median([1,1,4,4]))
10  #print(median(3))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
3
2.5
```

Gdy podamy w parametrze listę innych wartości np. char, to otrzymamy stosowny komunikat

```
11  print(median(['s','k']))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
Podana lista zawiera wartości niebędące int
None
```

Gdy nie podamy w parametrze listy, tylko np. pojedynczy char to otrzymamy stosowny komunikat

```
10  print(median('ch'))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
Parametr musi być listą liczb
None
```

c. pierwiastek

```
13
14  print(pierwiastek(3,0.001))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
1.7342174176198113
```

W przypadku podania jako parametr wartości nie liczbowej lub podania błędnego eplisonu otrzymujemy komunikaty:

```
14  #print(pierwiastek(3,0.001))
15  print(pierwiastek('a',0.001))
16  print(pierwiastek(3,-0.001))
17
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
Nie podano liczby jako parametr!
None
Podano ujemny epsilon
None
```

d. make_alpha dict

```
18  print(make_alpha_dict('on i ona'))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
{'a': ['ona'], 'i': ['i'], 'n': ['on', 'ona'], 'o': ['on', 'ona']}
```

W przypadku podania pustego ciągu znaków jako parametr lub innej wartości(np. int) użytkownik dostaje odpowiednie komunikaty:

```
17
18  #print(make_alpha_dict('on i ona'))
19  print(make_alpha_dict('   '))
20  print(make_alpha_dict(7))
21
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
Podany łańcuch nie zawiera słów!
None
Podany parametr nie jest ciągiem znaków!
None
```

e. flatten

```
21
22  print(flatten([1, [2, 3], [[4, 5], 6]]))
23
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
[1, 2, 3, 4, 5, 6]
```

Zadanie 2

a. for_all

```
25  print(for_all(lambda x: x%2==0,[4,7,2,6,8,8,2]))
26  print(for_all(lambda x: x%2==0,[4,2,2,6,8,8,2]))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
False
True
```

W przypadku gdy w parametrze nie podamy obiektu który jest typem Iterable to dostaniemy komunikat:

```
27  print(for_all(lambda x: x%2==0,4))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
Parametr nie jest typem iterable!
None
```

Taki sam komunikat wyświetla się dla reszty funkcji z zadania 2 gdy nie podamy typu iterable.

b. exists

```
29  print(exists(lambda x: x%2==0,[3,5,7,9,9]))
30  print(exists(lambda x: x%2==0,[3,5,2,9,9]))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
False
True
```

c. atleast

```
32  print(atleast(3, lambda x: x>0,[2,7,9,-8,-2]))
33  print(atleast(3, lambda x: x>0,[2,7,0,-8,-2]))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
True
False
```

d. atmost

```
35  print(atmost(3, lambda x: x>0,[2,7,9,-8,-2]))
36  print(atmost(3, lambda x: x>0,[2,7,-3,-8,-2]))
37  print(atmost(3, lambda x: x>0,[2,7,8,8,-2,2]))
38  print(atmost(3, lambda x: x>0,[0,-2,2]))
39
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python main.py
True
True
False
True
```

Zadanie 3

Iterator zwraca kolejne losowo generowane hasła.

Wyświetlanie kolejnych haseł za pomocą next:

```
27  password = PasswordGenerator(length=8, charset=string.ascii_lowercase, count=4)
28
29
30  print(next(password))
31  print(next(password))
32  print(next(password))
33  print(next(password))
34  print(next(password))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python zad3.py
ovrsqnjh
cpgnanrd
ymbahaoi
fekwaacx
Traceback (most recent call last):
  File "/home/student/Pulpit/Lab7/zad3.py", line 34, in <module>
    print(next(password))
  File "/home/student/Pulpit/Lab7/zad3.py", line 17, in __next__
    raise StopIteration
StopIteration
```

Wyświetlanie kolejnych haseł w pętli for:

```
36  password2 = PasswordGenerator(length=8, count=4)
37
38  for passw in password2:
39      print(passw)
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python zad3.py
TrrMwfc2
SRuSvhXB
hBpnySY0
4gMH8jqg
```

Zadanie 4

Funkcja make_generator zwracającą generator.

```
19  fib_generator = make_generator(fib)
20  print("Ciąg Fibonacciego")
21  for i in range(10):
22      print(next(fib_generator))
23
24  geo_generator = make_generator(lambda n: 2 ** n)
25  print("Ciąg geometryczny")
26  for i in range(10):
27      print(next(geo_generator))
28
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python zad4.py
Ciąg Fibonacciego
1
1
2
3
5
8
13
21
34
55
Ciąg geometryczny
2
4
8
16
32
64
128
256
512
1024
```

Zadanie 5

Funkcja make_generator_mem, która działa tak,jak make_generator, ale memoizuje funkcję f.

```python
fib_generator = make_generator_mem(fib)
print("Ciąg Fibonacciego zad5")
for i in range(10):
    print(next(fib_generator))


geo_generator = make_generator_mem(lambda n: 2 ** n)
print("Ciąg geometryczny zad5")
for i in range(10):
    print(next(geo_generator))
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python zad5.py
Ciąg Fibonacciego zad5
1
1
2
3
5
8
13
21
34
55
Ciąg geometryczny zad5
2
4
8
16
32
64
128
256
512
1024
```

Zadanie 6

Dekorator log, który służy do dekorowania funkcji lub klas.

Zad6_test.py:

```
18
19  my_function(2,4)
20  my_function(3,4)
21  my_function1(3,4)
22
23  test= SampleClass(5)
24
```

```
student@jezyki-skryptowe:~/Pulpit/Lab7$ python zad6_test.py
2023-04-25 13:27:16,800 INFO Nazwa funkcji: my_function  Argumenty wywołania: ((2, 4))  Wartość zwracana: 6  Czas wywołania:  2023-04-25 13:27:16.800807   Czas trwania: 2.384185791015625e-07s
2023-04-25 13:27:16,801 INFO Nazwa funkcji: my_function  Argumenty wywołania: ((3, 4))  Wartość zwracana: 7  Czas wywołania:  2023-04-25 13:27:16.801064   Czas trwania: 2.384185791015625e-07s
2023-04-25 13:27:16,801 DEBUG Nazwa funkcji: my_function1  Argumenty wywołania: ((3, 4))  Wartość zwracana: -1  Czas wywołania:  2023-04-25 13:27:16.801168   Czas trwania: 0.0s
2023-04-25 13:27:16,801 DEBUG Nazwa funkcji: __init__  Argumenty wywołania: ((<__main__.SampleClass object at 0x7f57ae76eda0>, 5))  Wartość zwracana: None  Czas wywołania:  2023-04-25 13:27:16.
801256   Czas trwania: 2.384185791015625e-07s
```