,

Laboratorium 2

Analiza Danych Pomiarowych
Ćwiczenia Laboratoryjne
Inżynieria Biomedyczna 2019/2020

Zagadnienia

Zagadnienia do samodzielnego opracowania:

- Klasy, metody.
- Przeciążanie operatorów.
- Iteratory.
- Generatory.
- Funkcje wewnętrzne.
- Dekoratory.
- Podstawowa obsługa wyjątków.

Zadania

Zadanie 1.
Zdefiniuj klasę **TempMeasurement** zawierającą składowe **temp_1** oraz **temp_2** inicjalizowane w konstruktorze. Klasa powinna również zawierać metodę **temp_combinations**, która zwróci listę krotek wszystkich kombinacji dla dwóch zakresów zamkniętych: od 0 do pierwszej składowej i od 0 do drugiej składowej.
Przykład i przewidywany rezultat:

```
my_operations = TempMeasurement(2, 3)
print(my_operations.temp_combinations())
>>> [(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2),
(1, 3), (2, 0), (2, 1), (2, 2), (2, 3)]
```

Zadanie 2.
Zdefiniuj klasę **Measurement** zawierającą składowe **p1** oraz **p2**. Przeciąż funkcje odpowiedzialną za konwersje do stringa oraz reprezentacje w taki sposób, aby funkcja **print** wypisywała p1 i p2 oddzielone spacją.
Przykład i przewidywany rezultat:

```
meas_1 = Measurement("Pomiar", "Temperatury")
meas_2 = Measurement("Pomiar", "Rezystancji")
meas_3 = Measurement("Pomiar", "pH")

meas_list = [meas_1, meas_2, meas_3]
print(meas_list)
>>> [Pomiar Temperatury, Pomiar Rezystancji, Pomiar pH]
```

Zadanie 3.
Zdefiniuj klasę **Counter** zawierającą trzy składowe: **start, stop, step**. Zachowanie klasy powinno być analogiczne do zachowania funkcji **range** z tym, że klasa **Counter** powinna być bezpośrednim iteratorem. Do definicji klasy nie wykorzystuj funkcji **range** ani jej odpowiedników.
Przykład i przewidywany rezultat:

```
my_counter = Counter(start=10, stop=20, step=2)
for value in my_counter:
        print(value)
>>> 10, 12, 14, 16, 18
```

Zadanie 4.
Zdefiniuj funkcję **my_generator** będącą generatorem o działaniu analogicznym do klasy **Counter** z zadania poprzedniego. W tym przypadku również nie wykorzystuj funkcji **range** ani jej odpowiedników.
Przykład i przewidywany rezultat:

```
my_gen = my_generator(start=10, stop=20, step=2)
for value in my_gen:
        print(value)
>>> 10, 12, 14, 16, 18
```

Zadanie 5.
Zdefiniuj funkcję **my_multiply** przyjmującej jako argument liczbę. Funkcja powinna zwrócić funkcję, której zakładanym argumentem jest lista, które z kolei przemnoży (i zmieni) wszystkie elementy listy wejściowej przez argument funkcji **my_multiply**.
Przykład i przewidywany rezultat:

```
example_func = my_multiply(5.5)
example_list = [1, 2, 3, 4]
print(example_list)
example_func(example_list)
print(example_list)
>>> [1, 2, 3, 4]
>>> [5.5, 11.0, 16.5, 22.0]
```

Zadanie 6.
Zdefiniuj funkcję **multiply_my** przyjmującej jako argument listę. Funkcja powinna zwrócić funkcję przyjmującą jako argument liczbę, która zwróci nową listę, stanowiącą pierwotną listę funkcji z każdym argumentem przemnożonym przez zadaną liczbę.
Przykład i przewidywany rezultat:

```
example_list = [1, 2, 3, 4]
example_func = multiply_my(example_list)
print(example_list)
print(example_func(5.5))
print(example_list)
>>> [1, 2, 3, 4]
>>> [5.5, 11.0, 16.5, 22.0]
>>> [1, 2, 3, 4]
```

Zadanie 7.
Zdefiniuj funkcję **try_except_ex**, która jako argument wejściowy przyjmie słownik oraz listę. Funkcja powinna sprawdzić czy słownik zawiera zdefiniowane klucze dla każdego elementu listy. W przeciwnym przypadku powinna zgłosić błąd **ValueError**.
Przykład i przewidywany rezultat:

```
input_dict = {'a': 5, 'b': 7, 'c': 4}
input_list_1 = ['a', 'b', 'c']
input_list_2 = ['a', 'b', 'd']

try:
        try_except_ex(input_dict, input_list_1)
        print("Ok")
except ValueError:
        print("Value␣Error")
```

```
try:
        try_except_ex(input_dict, input_list_2)
        print("Ok")
except ValueError:
        print("Value Error")
```

```
>>> Ok
>>> Value Error
```

Zadanie 8 dodatkowe

Zdefiniuj klasę **VectorExample** z składową **vector**, z przeciążonymi operatorami dodawania, odejmowania, mnożenia, dzielenia, reprezentacji łańcucha. W przypadku dzielenia, jeżeli element będący dzielnikiem jest zerem metoda powinna zgłosić wyjątek **ZeroDivisionError**. W przypadku różnej długości wektorów powinien zostać zgłoszony wyjątek **ValueError** z komentarzem dotyczącym różnej długości wektorów. Sprawdzanie długości powinno odbywać się w dekoratorze.

Przykład i przewidywany rezultat:

```
vector_1 = VectorExample([1, 2, 3, 4])
vector_2 = VectorExample([5, 3, 2, 1])
vector_3 = VectorExample([2, 3, 1])
vector_4 = VectorExample([7, 0, 4, 2])

print(vector_1 + vector_2)
print(vector_1 - vector_2)
print(vector_1 * vector_2)
print(vector_1 / vector_2)

try:
        print(vector_1 + vector_3)
except Exception as e:
        print(e)

try:
        print(vector_1 / vector_4)
except Exception as e:
        print(e)
```

```
>>> [6, 5, 5, 5]
>>> [-4, -1, 1, 3]
>>> [5, 6, 6, 4]
>>> [0.2, 0.6666666666666666, 1.5, 4.0]
>>> Vectors must be the same length.
>>> Divided by zero.
```