Wprowadzenie do języka Python - Wykład 6

Moduł collections

Moduł collections dostarcza specjalizowanych kontenerów typu słownik, listy, zbiorów i krotki. Zawiera alternatywy dla standardowych kontenerów, które oferują dodatkowe funkcjonalności i wydajność. Niektóre z najbardziej znanych i użytecznych struktur danych w module collections to:

- `namedtuple`: Pozwala tworzyć prostą "strukturę", która składa się z atrybutów, z zachowaniem małego zużycia pamięci. Ułatwia zrozumienie kodu, ponieważ atrybuty mają nazwy, w przeciwieństwie do standardowych krotek.
- `deque`: Dwukierunkowa kolejka (double-ended queue), która umożliwia szybkie dodawanie i usuwanie elementów z obu końców kolekcji. Jest to szczególnie użyteczne w algorytmach wymagających operacji na początku i końcu sekwencji.

- `Counter`: Klasa, która pozwala na zliczanie elementów w kolekcji (np. listy, słowniki, napisy). Może być używana do łatwego obliczania statystyk, takich jak częstotliwość występowania elementów.
- `OrderedDict`: Słownik, który zachowuje kolejność, w jakiej elementy zostały dodane. Jest to przydatne, gdy kolejność elementów ma znaczenie.
- `defaultdict`: Specjalny rodzaj słownika, który automatycznie tworzy wartość domyślną dla klucza, którego nie ma w słowniku. Ułatwia to pracę z danymi, gdy wartość domyślna dla nieistniejącego klucza jest wymagana.

nametuple

Inspiracja:

```
# Create a 2D point as a tuple
point = (2, 4)
print(point)
# Access coordinate x
print(point[0])
# Access coordinate y
print(point[1])
# Try to update a coordinate value
# point[0] = 3
```

```
(2, 4)
2
4
```

```python
from collections import namedtuple

# Create a namedtuple type, Point
Point = namedtuple("Point", "x y")
issubclass(Point, tuple)
# Instantiate the new type
point = Point(2, 4)
print(point)
# Dot notation to access coordinates
print(point.x)
print(point.y)
# Indexing to access coordinates
print(point[0])
print(point[1])
# Named tuples are immutable
# point.x = 100
```

```
Point(x=2, y=4)
2
4
2
4
```

Szczegóły:

https://docs.python.org/3/library/collections.html#collections.nar

```python
from collections import namedtuple

Osoba = namedtuple('Osoba', ['imie', 'wiek', 'miasto'])
osoba1 = Osoba('Anna', 28, 'Warszawa')
osoba2 = Osoba('Jan', 35, 'Kraków')
print(osoba1.imie)  # Wypisze 'Anna'
print(osoba1.wiek)  # Wypisze 28
print(osoba1.miasto)  # Wypisze 'Warszawa'

print(osoba2.imie)  # Wypisze 'Jan'
print(osoba2.wiek)  # Wypisze 35
print(osoba2.miasto)  # Wypisze 'Kraków'
```

```
Anna
28
Warszawa
Jan
35
Kraków
```

`_make(iterable)`: Tworzy nową instancję namedtuple na podstawie wartości z przekazanego obiektu iterowalnego, takiego jak lista lub inna krotka.

```
1  from collections import namedtuple
2
3  Osoba = namedtuple('Osoba', ['imie', 'wiek', 'miasto'])
4  dane = ['Katarzyna', 32, 'Poznań']
5  osoba3 = Osoba._make(dane)
6  print(osoba3)
```

```
Osoba(imie='Katarzyna', wiek=32, miasto='Poznań')
```

`_asdict()`: Konwertuje instancję namedtuple na słownik, gdzie klucze to nazwy atrybutów, a wartości to ich wartości.

```
1 from collections import namedtuple
2
3 Osoba = namedtuple('Osoba', ['imie', 'wiek', 'miasto'])
4 dane = ['Katarzyna', 32, 'Poznań']
5 osoba3 = Osoba._make(dane)
6 osoba3_dict = osoba3._asdict()
7 print(osoba3_dict)
```

```
{'imie': 'Katarzyna', 'wiek': 32, 'miasto': 'Poznań'}
```

`_replace(**kwargs)`: Tworzy nową instancję namedtuple, zastępując wartości niektórych atrybutów. Metoda przyjmuje argumenty słownikowe, gdzie klucze to nazwy atrybutów, a wartości to nowe wartości.

```
from collections import namedtuple

Osoba = namedtuple('Osoba', ['imie', 'wiek', 'miasto'])
dane = ['Katarzyna', 32, 'Poznań']
osoba3 = Osoba._make(dane)
osoba4 = osoba3._replace(wiek=33)
print(osoba4)
```

```
Osoba(imie='Katarzyna', wiek=33, miasto='Poznań')
```

`_fields`: Atrybut krotki, który zawiera wszystkie pola namedtuple w kolejności, w jakiej zostały zdefiniowane.

```
1 from collections import namedtuple
2
3 Osoba = namedtuple('Osoba', ['imie', 'wiek', 'miasto'])
4 print(Osoba._fields)
```

```
('imie', 'wiek', 'miasto')
```

`_field_defaults`: Atrybut słownika, który przechowuje wartości domyślne dla pól.

```
1 from collections import namedtuple
2
3 Osoba = namedtuple('Osoba', ['imie', 'wiek', 'miasto'], defaults=['Nieznane
4
5 osoba5 = Osoba('Marek')
6 print(osoba5)
7 print(Osoba._field_defaults)
```

```
Osoba(imie='Marek', wiek=None, miasto=None)
{'imie': 'Nieznane', 'wiek': None, 'miasto': None}
```

Wersja z typowaniem

https://mypy.readthedocs.io/en/stable/kinds_of_types.html#nam
tuples

Zastosowanie w geometrii

```
 1 from collections import namedtuple
 2
 3 Punkt2D = namedtuple('Punkt2D', ['x', 'y'])
 4 Punkt3D = namedtuple('Punkt3D', ['x', 'y', 'z'])
 5
 6 punkt2d = Punkt2D(3, 4)
 7 punkt3d = Punkt3D(1, 2, 3)
 8
 9 print(punkt2d)  # Wypisze "Punkt2D(x=3, y=4)"
10 print(punkt3d)  # Wypisze "Punkt3D(x=1, y=2, z=3)"
```

Zastosowanie w połączeniach z bazą danych

```python
from collections import namedtuple
import sqlite3

Osoba = namedtuple('Osoba', ['id', 'imie', 'nazwisko'])

# Przykład połączenia z bazą danych SQLite
conn = sqlite3.connect('przykladowa_baza.db')
cursor = conn.cursor()

cursor.execute("SELECT id, imie, nazwisko FROM osoby")

wyniki = cursor.fetchall()

osoby = [Osoba(*wynik) for wynik in wyniki]

for osoba in osoby:
    print(f"ID: {osoba.id}, Imię: {osoba.imie}, Nazwisko: {osoba.nazwisko}"
```

Kolory

```
 1 from collections import namedtuple
 2
 3 KolorRGB = namedtuple('KolorRGB', ['r', 'g', 'b'])
 4 KolorHSL = namedtuple('KolorHSL', ['h', 's', 'l'])
 5
 6 kolor_rgb = KolorRGB(255, 0, 0)
 7 kolor_hsl = KolorHSL(0, 100, 50)
 8
 9 print(kolor_rgb)  # Wypisze "KolorRGB(r=255, g=0, b=0)"
10 print(kolor_hsl)  # Wypisze "KolorHSL(h=0, s=100, l=50)"
```

Import z csv

```
 1 import csv
 2 from collections import namedtuple
 3
 4 with open('przykladowy_plik.csv', mode='r') as csvfile:
 5     csv_reader = csv.reader(csvfile)
 6     headers = next(csv_reader)
 7     Rekord = namedtuple('Rekord', headers)
 8
 9     for row in csv_reader:
10         rekord = Rekord(*row)
11         print(rekord)
```

deque

deque to skrót od “double-ended queue”. Działa jak dwukierunkowa kolejka, co oznacza, że można dodawać i usuwać elementy z obu końców kolekcji w sposób efektywny. deque jest szczególnie przydatne w sytuacjach, gdy potrzebujemy wykonywać operacje na początku i końcu sekwencji z wysoką wydajnością.

```python
from collections import deque

dq = deque()

# Dodawanie elementów na koniec kolejki
dq.append('B')
dq.append('C')

# Dodawanie elementów na początek kolejki
dq.appendleft('A')

print(dq)  # Wypisze "deque(['A', 'B', 'C'])"

# Usuwanie elementów z końca kolejki
element_koniec = dq.pop()
print(element_koniec)  # Wypisze "C"
print(dq)  # Wypisze "deque(['A', 'B'])"

# Usuwanie elementów z początku kolejki
```

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

```python
from collections import deque

dq = deque(['A', 'B', 'C', 'D'])

# Obracanie w prawo o 1 pozycję
dq.rotate(1)
print(dq)  # Wypisze "deque(['D', 'A', 'B', 'C'])"

# Obracanie w lewo o 2 pozycje
dq.rotate(-2)
print(dq)  # Wypisze "deque(['B', 'C', 'D', 'A'])"
```

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

```python
from collections import deque

dq_ograniczone = deque(maxlen=3)

dq_ograniczone.append(1)
dq_ograniczone.append(2)
dq_ograniczone.append(3)

print(dq_ograniczone)  # Wypisze "deque([1, 2, 3], maxlen=3)"

dq_ograniczone.append(4)  # Usunie element "1" z początku kolejki
print(dq_ograniczone)  # Wypisze "deque([2, 3, 4], maxlen=3)"
```

```
deque([1, 2, 3], maxlen=3)
deque([2, 3, 4], maxlen=3)
```

Szczegóły

https://docs.python.org/3/library/collections.html#deque-objects

Porównanie z listami

```python
from collections import deque
from time import perf_counter

TIMES = 10_000
a_list = []
a_deque = deque()


def average_time(func, times):
    total = 0.0
    for i in range(times):
        start = perf_counter()
        func(i)
        total += (perf_counter() - start) * 1e9
    return total / times


list_time = average_time(lambda i: a_list.insert(0, i), TIMES)
```

```
list.insert()      1557.78 ns
deque.appendleft() 124.79 ns  (12.4832x faster)
```

Counter

Counter to klasa, która pozwala na liczenie elementów w kolekcjach, takich jak listy, krotki czy napisy. Counter działa jak słownik, gdzie klucze to unikalne elementy kolekcji, a wartości to liczba wystąpień tych elementów.

```python
from collections import Counter

lista = ['A', 'B', 'A', 'C', 'A', 'B', 'A']
licznik = Counter(lista)

print(licznik)
```

```
Counter({'A': 4, 'B': 2, 'C': 1})
```

```python
from collections import Counter

napis = "przykladowy_napis"
licznik_napis = Counter(napis)

print(licznik_napis)
```

```
Counter({'p': 2, 'y': 2, 'a': 2, 'r': 1, 'z': 1, 'k': 1, 'l': 1, 'd': 1, 'o':
1, 'w': 1, '_': 1, 'n': 1, 'i': 1, 's': 1})
```

```python
from collections import Counter

lista = ['A', 'B', 'A', 'C', 'A', 'B', 'A']
licznik = Counter(lista)
licznik.update(['B', 'C', 'C', 'D'])

print(licznik)
```

```
Counter({'A': 4, 'B': 3, 'C': 3, 'D': 1})
```

```python
from collections import Counter

lista = ['A', 'B', 'A', 'C', 'A', 'B', 'A']
licznik = Counter(lista)
licznik.update(['B', 'C', 'C', 'D'])
najczestsze = licznik.most_common(2)

print(najczestsze)
```

```
[('A', 4), ('B', 3)]
```

```python
from collections import Counter

licznik1 = Counter(a=3, b=2, c=1)
licznik2 = Counter(a=1, b=2, c=3)

licznik1.subtract(licznik2)

print(licznik1)
```

```
Counter({'a': 2, 'b': 0, 'c': -2})
```

```python
from collections import Counter

lista = ['A', 'B', 'A', 'C', 'A', 'B', 'A']
licznik = Counter(lista)
licznik.update(['B', 'C', 'C', 'D'])
suma = sum(licznik.values())

print(suma)
```

```
11
```

Szczegóły:

https://docs.python.org/3/library/collections.html#counter-objects

Zliczanie słów

```python
from collections import Counter
import re

tekst = """
Python to język programowania wysokiego poziomu ogólnego przeznaczenia.
Python jest prosty w nauce i czytaniu. Python jest popularny w różnych dzie
"""

# Usuwanie znaków interpunkcyjnych i konwersja na małe litery
oczyszczony_tekst = re.sub(r'[^\w\s]', '', tekst.lower())

# Podział tekstu na słowa
slowa = oczyszczony_tekst.split()

# Zliczenie wystąpień słów
licznik_slow = Counter(slowa)

print(licznik_slow)
```

```
Counter({'python': 3, 'jest': 2, 'w': 2, 'to': 1, 'język': 1, 'programowania':
1, 'wysokiego': 1, 'poziomu': 1, 'ogólnego': 1, 'przeznaczenia': 1, 'prosty':
1, 'nauce': 1, 'i': 1, 'czytaniu': 1, 'popularny': 1, 'różnych': 1,
'dziedzinach': 1})
```

Zliczanie elementów w liście

```
1 from collections import Counter
2
3 lista_elementow = ['A', 'B', 'C', 'A', 'B', 'A', 'D', 'A', 'C', 'B', 'C', '
4
5 licznik_elementow = Counter(lista_elementow)
6
7 print(licznik_elementow)
```

```
Counter({'A': 5, 'B': 3, 'C': 3, 'D': 1})
```

Zliczanie znaków w napise

```python
from collections import Counter

napis = "Przykladowy napis zliczajacy wystapienia znakow"

licznik_znakow = Counter(napis.lower())

print(licznik_znakow)
```

```
Counter({'a': 7, 'z': 4, 'y': 4, ' ': 4, 'i': 4, 'p': 3, 'w': 3, 'n': 3, 'k': 2, 'l': 2, 'o': 2, 's': 2, 'c': 2, 'r': 1, 'd': 1, 'j': 1, 't': 1, 'e': 1})
```

Zliczanie elementów w dwóch listach

```
 1 from collections import Counter
 2
 3 lista1 = ['A', 'B', 'C', 'A', 'B', 'A']
 4 lista2 = ['B', 'C', 'D', 'C', 'A', 'B', 'A']
 5
 6 licznik_lista1 = Counter(lista1)
 7 licznik_lista2 = Counter(lista2)
 8
 9 suma_licznikow = licznik_lista1 + licznik_lista2
10
11 print(suma_licznikow)
```

```
Counter({'A': 5, 'B': 4, 'C': 3, 'D': 1})
```

```python
from collections import Counter

# Fruit sold per day
sales_day1 = Counter(apple=4, orange=9, banana=4)
sales_day2 = Counter(apple=10, orange=8, banana=6)

# Total sales
print(sales_day1 + sales_day2)


# Sales increment
print(sales_day2 - sales_day1)


# Minimum sales
print(sales_day1 & sales_day2)


# Maximum sales
```

```
Counter({'orange': 17, 'apple': 14, 'banana': 10})
Counter({'apple': 6, 'banana': 2})
Counter({'orange': 8, 'apple': 4, 'banana': 4})
Counter({'apple': 10, 'orange': 9, 'banana': 6})
```

OrderedDict

`OrderedDict` to klasa, która działa jak zwykły słownik, ale zachowuje kolejność wstawiania elementów. Oznacza to, że podczas iteracji po OrderedDict, elementy są zwracane w kolejności, w której zostały dodane.

```
1 from collections import OrderedDict
2
3 slownik_uporzadkowany = OrderedDict()
4
5 slownik_uporzadkowany['A'] = 1
6 slownik_uporzadkowany['B'] = 2
7 slownik_uporzadkowany['C'] = 3
8
9 print(slownik_uporzadkowany)
```

```
OrderedDict([('A', 1), ('B', 2), ('C', 3)])
```

```python
from collections import OrderedDict

zwykly_slownik = {'C': 3, 'A': 1, 'B': 2}

slownik_uporzadkowany = OrderedDict(sorted(zwykly_slownik.items()))

print(slownik_uporzadkowany)
```

```
OrderedDict([('A', 1), ('B', 2), ('C', 3)])
```

```python
from collections import OrderedDict

zwykly_slownik = {'C': 3, 'A': 1, 'B': 2}

slownik_uporzadkowany = OrderedDict(sorted(zwykly_slownik.items()))

slownik_uporzadkowany.pop('B')
slownik_uporzadkowany['B'] = 2

print(slownik_uporzadkowany)

for klucz, wartosc in slownik_uporzadkowany.items():
    print(f"{klucz}: {wartosc}")
```

```
OrderedDict([('A', 1), ('C', 3), ('B', 2)])
A: 1
C: 3
B: 2
```

Szczegóły:

https://docs.python.org/3/library/collections.html#ordereddict-objects

defaultdict

`defaultdict` to klasa, która działa jak zwykły słownik, ale posiada domyślną wartość dla nieistniejących kluczy. Przy tworzeniu defaultdict musimy podać funkcję, która będzie zwracać domyślną wartość dla nieistniejących kluczy.

```
1 from collections import defaultdict
2
3 slownik_licznik = defaultdict(int)
4
5 slownik_licznik['A'] += 1
6 slownik_licznik['B'] += 1
7 slownik_licznik['A'] += 1
8
9 print(slownik_licznik)
```

```
defaultdict(<class 'int'>, {'A': 2, 'B': 1})
```

```python
from collections import defaultdict

slownik_list = defaultdict(list)

slownik_list['A'].append(1)
slownik_list['B'].append(2)
slownik_list['A'].append(3)

print(slownik_list)
```

```
defaultdict(<class 'list'>, {'A': [1, 3], 'B': [2]})
```

```python
from collections import defaultdict

slownik_zbior = defaultdict(set)

slownik_zbior['A'].add(1)
slownik_zbior['B'].add(2)
slownik_zbior['A'].add(3)
slownik_zbior['A'].add(1)

print(slownik_zbior)
```

```
defaultdict(<class 'set'>, {'A': {1, 3}, 'B': {2}})
```

```python
from collections import defaultdict

def domyslna_wartosc():
    return "Nieznany"

slownik_wlasny = defaultdict(domyslna_wartosc)

slownik_wlasny['A'] = 'Kot'
slownik_wlasny['B'] = 'Pies'

print(slownik_wlasny['A'])
print(slownik_wlasny['C'])
```

```
Kot
Nieznany
```

Zastosowania

```
1 from collections import defaultdict
2
3 lista_elementow = ['A', 'B', 'C', 'A', 'B', 'A', 'D', 'A', 'C', 'B', 'C', '
4 licznik_elementow = defaultdict(int)
5
6 for element in lista_elementow:
7     licznik_elementow[element] += 1
8
9 print(licznik_elementow)
```

```
defaultdict(<class 'int'>, {'A': 5, 'B': 3, 'C': 3, 'D': 1})
```

```python
from collections import defaultdict

tekst = "przykładowy tekst do zliczenia wystąpień słów w tekście"
slowa = tekst.split()

licznik_slow = defaultdict(int)

for slowo in slowa:
    licznik_slow[slowo] += 1

print(licznik_slow)
```

```
defaultdict(<class 'int'>, {'przykładowy': 1, 'tekst': 1, 'do': 1,
'zliczenia': 1, 'wystąpień': 1, 'słów': 1, 'w': 1, 'tekście': 1})
```

```python
from collections import defaultdict

lista = ['A', 'B', 'C', 'A', 'B', 'A', 'D', 'A', 'C', 'B', 'C', 'A']
indeksy = defaultdict(list)

for i, element in enumerate(lista):
    indeksy[element].append(i)

print(indeksy)
```

```
defaultdict(<class 'list'>, {'A': [0, 3, 5, 7, 11], 'B': [1, 4, 9], 'C': [2,
8, 10], 'D': [6]})
```

ChainMap

ChainMap to klasa z modułu collections w Pythonie, która pozwala na połączenie wielu słowników w jedną strukturę. Działa jako “widok” na złączonych słownikach, co oznacza, że nie tworzy nowego słownika, lecz pozwala na manipulowanie istniejącymi słownikami jako jednym obiektem.

```python
from collections import ChainMap

slownik1 = {'A': 1, 'B': 2, 'C': 3}
slownik2 = {'D': 4, 'E': 5, 'F': 6}

polaczony_slownik = ChainMap(slownik1, slownik2)

print(polaczony_slownik)
print(polaczony_slownik['A'])
print(polaczony_slownik['D'])
```

```
ChainMap({'A': 1, 'B': 2, 'C': 3}, {'D': 4, 'E': 5, 'F': 6})
1
4
```

```python
from collections import ChainMap

slownik1 = {'A': 1, 'B': 2, 'C': 3}
slownik2 = {'D': 4, 'A': 5, 'F': 6}

polaczony_slownik = ChainMap(slownik1, slownik2)

print(polaczony_slownik)
print(polaczony_slownik['A'])
print(polaczony_slownik['D'])
```

```
ChainMap({'A': 1, 'B': 2, 'C': 3}, {'D': 4, 'A': 5, 'F': 6})
1
4
```

```python
from collections import ChainMap

slownik1 = {'A': 1, 'B': 2, 'C': 3}
slownik2 = {'D': 4, 'E': 5, 'F': 6}

polaczony_slownik = ChainMap(slownik1, slownik2)
slownik3 = {'G': 7, 'H': 8, 'I': 9}

polaczony_slownik = polaczony_slownik.new_child(slownik3)

print(polaczony_slownik)

for klucz, wartosc in polaczony_slownik.items():
    print(f"{klucz}: {wartosc}")
```

```
ChainMap({'G': 7, 'H': 8, 'I': 9}, {'A': 1, 'B': 2, 'C': 3}, {'D': 4, 'E': 5,
'F': 6})
D: 4
E: 5
F: 6
A: 1
B: 2
C: 3
G: 7
H: 8
I: 9
```

Bibliografia

- https://docs.python.org/3/library/collections.html#module-collections, dostęp online 2.04.2023.
- https://realpython.com/python-namedtuple/, dostęp online 2.04.2023.
- https://realpython.com/python-deque/, dostęp online 2.04.2023.
- https://realpython.com/python-counter/, dostęp online 2.04.2023.