Laboratorium 3

Typy złożone w języku Python oraz ich rozszerzenia, moduł collections oraz dataclasses

Zagadnienia:

- Typy złożone w języku Python (list, dict, tuple, set)
- Typ NamedTuple (collections)
- Typ DataClass (dataclasses)
- Typ DefaultDict (collections)
- Typ Deque (collections)
- Typ Counter (collections)

Wprowadzenie

W języku Python, typy złożone są niezwykle istotnym elementem, umożliwiając programistom efektywne zarządzanie bardziej zaawansowanymi strukturami danych. Dzięki nim, programiści mogą tworzyć bardziej złożone i elastyczne rozwiązania, które sprostają różnorodnym wymaganiom projektowym. Typy złożone stanowią fundament wielu operacji przetwarzania danych, a ich zastosowanie jest kluczowe zarówno w analizie danych, jak i w projektowaniu oprogramowania.

Poniżej znajduje się tabela podsumowująca właściwości wbudowanych typów złożonych:

	Lista (list)	Słownik (dict)	Zbiór (set)	Krotka (tuple)
Dostęp	Indeks	Klucz	Brak	Indeks
Kolejność	Tak	Tak	Nie	Tak
Duplikaty	Tak	Nie	Nie	Tak
Zmienność	Tak	Tak	Tak	Nie

Typ namedtuple (collections)

Typ ten pozwala na tworzenie podklas krotek z nazwanymi polami. Pola te dają bezpośredni dostęp do wartości w danej nazwanej krotce przy użyciu operatora kropki (analogicznie jak dostęp do atrybutów obiektu). Potrzeba tej funkcji powstała, ponieważ używanie indeksów do uzyskiwania dostępu do wartości w zwykłej krotce jest nieintuicyjne, trudne do odczytania i podatne na błędy. Jest to szczególnie istotne, gdy krotka ma kilka elementów i jest tworzona daleko od miejsca, w którym jest używana.

Rozważmy przykład funkcji obliczającej sumę i iloczyn dwóch liczb:

```
def sum_multi(x,y):
    return x+y,x*y

print(type(sum_multi(2,3)))
print(sum_multi(2,3))
```

Jak widać, wynik działania funkcji to krotka składająca się z dwóch liczb, jednak otrzymując taki wynik nie jesteśmy w stanie jednoznacznie zinterpretować znaczenia poszczególnych elementów krotki, więc

czytelność kodu jest ograniczona. Powyższą funkcję można zmodyfikować korzystając z typu NamedTuple w następujący sposób:

```
from collections import namedtuple
def sum_multi(x,y):
    SumMulti = namedtuple("SumMulti",["suma","iloczyn"])
    return SumMulti(x+y,x*y)

result=sum_multi(2,3)
print(type(result))
print(result)
print(result.suma)
print(result.iloczyn)
```

Tym razem widoczne jest znaczenie każdego elementu krotki wynikowej. Dodatkowo możliwy jest dostęp do poszczególnych jej elementów po ich nazwie. Nazwy elementów krotki można określać na różne sposoby:

1. Jako napis z nazwami oddzielonymi znakiem białym
2. Jako napis z nazwami oddzielonymi przecinkiem
3. Jako obiekt typu iterable (np. lista) z nazwami

```
Person = namedtuple("Person","name job age")
Person = namedtuple("Person","name,job,age")
Person = namedtuple("Person",["name", "job", "age"])
```

Typ NamedTuple posiada również kilka ciekawych funkcjonalności takich jak definiowanie wartości domyślnych dla elementów czy możliwość przekształcania w słownik:

```
Point = namedtuple("Point",["x","y"],defaults=[0,0])
point1 = Point()
print(point1)
dictpoint = point1._asdict()
print(dictpoint)
```

Typ dataclass (dataclasses)

Kolejnym typem zbliżonym do NamedTuple jest typ DataClass z modułu dataclasses. Typ ten używany jest jako dekorator klasy przechowującej dane (Data Class). Dekoratory w języku Python to zaawansowany mechanizm, który pozwala na modyfikowanie funkcji lub klas w sposób elastyczny i przejrzysty. Służą one do dodawania dodatkowej funkcjonalności do istniejących funkcji lub klas bez konieczności ich zmieniania. Dekoratory są szczególnie przydatne w przypadku powtarzających się operacji. Przykład zastosowania typu DataClass przedstawiono poniżej.

```
from dataclasses import dataclass
@dataclass
class Point:
    x:int
    y:int
```

```
point1=Point(2,3)
print(point1)
print(point1==Point(2,3))
```

Typ DataClass umożliwia szybkie tworzenie klas z prostymi funkcjonalnościami charakterystycznymi dla klas przechowujących dane. DataClass wspomaga również Type Annotation, wartości domyślne oraz umożliwia pisanie dodatkowych metod:

```
@dataclass
class Point:
    x: int = 0
    y: int = 0

    def isCenter(self) -> bool:
        return self.x == 0 and self.y == 0

p1 = Point()
p2 = Point(2,5)
print(p1.isCenter())
print(p2.isCenter())
```

Typ defaultdict (collections)

Typ defaultdict to podklasa słownika (dict), która automatycznie przypisuje wartości domyślne dla brakujących kluczy. Typ ten jako argument przyjmuje funkcję bezargumentową, która jest wywoływana w przypadku próby dostępu do nieistniejącego klucza:

```
from collections import defaultdict
def default_value():
    return "default"


my_dict = defaultdict(default_value)
my_dict["name"] = "Alice"
print(my_dict["name"])
print(my_dict["age"])
```

Wartością domyślną może być także typ danych lub kontener:

```
int_dict = defaultdict(int)
int_dict["count"] += 1

print(int_dict["count"])
print(int_dict["missing"])


list_dict = defaultdict(list)
list_dict["fruits"].append("apple")

print(list_dict["fruits"])
print(list_dict["vegies"])
```

Typ deque (collections)

Deque jest to sekwencyjny typ danych, który stanowi uogólnienie stosów i kolejek, zaprojektowany do obsługi wydajnych operacji dodawania i usuwanie elementów po obu stronach tego kontenera. W Pythonie operacje append i pop na początku lub po lewej stronie obiektów listy są nieefektywne i mają złożoność czasową O(n). Operacje te są szczególnie kosztowne, jeśli pracujemy z dużymi listami, ponieważ Python musi przesunąć wszystkie elementy w prawo, aby wstawić nowe elementy na początku listy. Z drugiej strony, operacje append i pop po prawej stronie listy są zwykle wydajne (O(1)), z wyjątkiem tych przypadków, w których Python musi ponownie przydzielić pamięć, aby zwiększyć bazową listę w celu zaakceptowania nowych elementów. Deque Pythona zostało stworzone w celu przezwyciężenia tego problemu. Operacje append i pop po obu stronach obiektu deque są stabilne i równie wydajne. Konstruktor obiektu Deque przyjmuje dwa argumenty:

1. Obiekt iterable, który ma zostać przekształcony na kolejkę dwustronną.

2. Wartość maxlen, czyli maksymalną długość kolejki (przekroczenie maksymalnej wielkości powoduje przesunięcie kolejki i usunięcie ostatniego elementu widzianego od strony dodania).

```
from collections import deque
recent_searches = deque(["programowanie python", "python", "python docs"],
maxlen=4)
print(recent_searches)
recent_searches.appendleft("politechnika lubelska")
print(recent_searches)
recent_searches.append("pollub plan zajec")
print(recent_searches)
```

Wiele metod charakterystycznych dla list jest również dostępna dla typu Deque. Warto jednak pamiętać, że:

- Kolejka dwustronna nie umożliwia wyciągnięcia elementu z dowolnego miejsca (tylko początek lub koniec) .
- Kolejka dwustronna może być obracana metodą rotate o dowolną liczbę miejsc .
- Kolejka dwustronna umożliwia dostęp do elementów po indeksach ale nie obsługuje fragmentowania w postaci [:2] itp.

Typ counter (collections)

Counter to specjalny typ danych w module collections w Pythonie, który służy do zliczania elementów w typach iterowanych (listy, ciągi znaków, krotki). Jest to podklasa słownika (dict), gdzie klucze są elementami, a wartościami są liczby wystąpień:

```
from collections import Counter
count = Counter(["apple","banana","banana","apple","apple"])
print(count)
count = Counter("abracadabra")
print(count.most_common(3))
```

Na obiektach typu Counter można też wykonywać operacje arytmetyczne, takie jak dodawanie, odejmowanie, przecięcie oraz suma:

```
print(count1 + count2)
print(count1 - count2)
print(count1 & count2)
print(count1 | count2)
```

Zadania

1. Zdefiniuj namedtuple o nazwie Product, która będzie reprezentowała produkty spożywcze z polami: "name", "price", "weight". Utwórz listę kilku produktów (przynajmniej 4) i wydrukuj na ekran. Następnie wydrukuj tylko nazwy wszystkich produktów.
2. Wykonaj polecenia zadania 1 ale zastąp typ namedtuple, typem dataclass.
3. Wykorzystaj typ defaultdict i zaimplementuj funkcję, która jako argument przyjmuje listę produktów (przynajmniej 10) z zadania 1 (namedtuple) i umożliwia ich grupowanie ze względu na nazwę ("name").
4. Korzystając z dataclass opracuj klasę reprezentującą pacjentów w kolejce do lekarza z polami "name", "surname", "age", "disease" oraz "visit_time" (godzina rejestracji – ten sam rok, miesiąc i dzień). Za pomocą typu deque utwórz kolejkę pacjentów przyjmowanych co 20 minut. Dodaj funkcję sortującą pacjentów (przynajmniej 5) pod względem czasu zaplanowanej wizyty i zwracającą posortowaną kolejkę. Przetestuj działanie kolejki dopóki ostatni pacjent nie zostanie przyjęty w kolejności godziny rejestracji, w przypadku gdy kolejka pozostanie pusta – wyświetl odpowiedni komunikat.
5. Wygeneruj listę 15 całkowitych liczb pseudolosowych w zakresie od 2 do 5 reprezentującą oceny na zaliczenie laboratorium. Policz liczbę wystąpień każdej oceny oraz 2 najczęściej występujące oceny. Napisz funkcję, która przyjmuje listę ocen i zwraca liczbę ocen pozytywnych.