PRZEGLĄD PODSTAWOWYCH BIBLIOTEK PYTHONA

ĆWICZENIA DODATKOWE

MODUŁ 8

Altkom Akademia S.A., materiały własne

1 MODUŁY I PAKIETY

ĆWICZENIE 1.1:

Implementacja iteratora

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętności:
  - implementacji własnych iteratorów
  - ich wykorzystania

CELE I ZADANIA:

- Utwórz klasę implementującą protokół iteratora
- Klasa powinna działać podobnie do funkcji *range*, ale argumentami nie są liczby całkowite, ale daty (obiekty typu *datetime.date*)
- Zadaniem iteratora jest zwracanie kolejnych dat z zadanego przedziału z podanym interwałem

ALGORYTM WYKONANIA:

- Utwórz klasę o nazwie *DateRange*, pełniącą rolę iteratora po datach
- Zadaniem iteratora jest zwracanie kolejnych dat z podanego przedziału, z określonym interwałem
- Tworząc iterator należy podać 3 argumenty:
  - datę początkową – parametr opcjonalny, domyślną wartością jest dzień dzisiejszy (obiekt typu *datetime.date*)
  - datę końcową – parametr obowiązkowy (obiekt typu *datetime.date*)
  - interwał – parametr opcjonalny, domyślną wartością jest 1 dzień (obiekt typu *int*)
- Zdefiniuj metodę *__init__* i przekaż do niej wartości definiujące przedział dat i interwał
- Metoda powinna zdefiniować atrybuty instancyjne przechowujące:
  - zakres dat
  - interwał
  - datę bieżącą
- Utwórz metodę *__iter__* zwracającą obiekt iteratora (może to być bieżąca instancja klasy)
- Utwórz metodę *__next__* odpowiedzialną za zwrócenie kolejnej daty
- Do modyfikacji bieżącej daty o wartość interwału możesz użyć obiektu *datetime.timedelta*
- Pamiętaj, aby zakończyć działanie iteratora wyrzuceniem wyjątku *StopIteration*
- Przetestuj działanie iteratora dla różnych zakresów dat

ĆWICZENIE 1.2:

Implementacja generatora

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętności:
  - tworzenia funkcji generatorów
  - wykorzystania generatorów do implementacji iteratorów

CELE I ZADANIA:

- Wzorując się na rozwiązaniu poprzedniego ćwiczenia 1.1, utwórz iterator dat z określonego przedziału za pomocą funkcji generatora

ALGORYTM WYKONANIA:

- Utwórz funkcję *date_generator* z dwoma parametrami definującymi przedział dat oraz jednym określającym interwał
  - data początkowa jest opcjonalna – domyślna wartość to dzień dzisiejszy
  - data końcowa jest obowiązkowa
  - interwał określający odstęp pomiędzy kolejnymi zwracanymi datami

  dwukrotnie
- Do zwrócenia wartości z funkcji generatora użyj polecenia `yield`
- Przetestuj działanie generatora dla różnych przedziałów

ĆWICZENIE 1.3:

Specjalizacja funkcji

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność tworzenia funkcji specjalizowanych na podstawie funkcji ogólnej

CELE I ZADANIA:

- Utwórz funkcję ogólną umożliwiającą zapis liczby dziesiętnej w dowolnym układzie pozycyjnym (o podstawie $\leqslant 36$)
- W oparciu o tę funkcję wyprowadź funkcje szczegółowe konwertujące liczbę dziesiętną na zapis w ustalonym układzie pozycyjnym

ALGORYTM WYKONANIA:

- Utwórz funkcję *decimal_to_radix* z dwoma parametrami reprezentującymi:
  - dziesiętną liczbę całkowitą do konwersji
  - podstawę układu pozycyjnego
- Zwracaną wartością jest zapis (w postaci tekstu) liczby w podanym układzie pozycyjnym
- Do zapisu cyfr liczby w danym układzie użyj symboli: cyfr (0-9) oraz liter (A-Z)
- Do wyodrębniania kolejnych cyfr możesz użyć funkcji *divmod*
- Na podstawie funkcji *decimal_to_radix* utwórz funkcje specjalizowane:
  - funkcję *dec_to_bin* zwracającą reprezentację liczby w systemie dwójkowym
  - funkcję *dec_to_oct* zwracającą reprezentację liczby w systemie ósemkowym
  - funkcję *dec_to_hex* zwracającą reprezentację liczby w systemie szesnastkowym
- W tym celu użyj funkcji *partial* z modułu *functools*
- Przetestuj działanie funkcji specjalizowanych

ĆWICZENIE 1.4:

Sortowanie i funkcja klucza

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność:
  - tworzenia funkcji klucza w sposób bezpośredni
  - konwersji funkcji komparatora na funkcję klucza
  - wykorzystania funkcje klucza do sortowania

CELE I ZADANIA:

- Utwórz prostą klasę stanową i kilka jej instancji
- Utwórz listę zawierającą te instancje
- Posortuj dane w liście w oparciu o zdefiniowaną funkcję klucza
- Utwórz analogiczną funkcję komparatora, a następnie skonwertuj ją na funkcję klucza wykorzystując odpowiedni dekorator z modułu *functools*
- Ponownie posortuj dane i porównaj wyniki

ALGORYTM WYKONANIA:

- Utwórz prostą klasę zawierającą dane – przykładowo może to być klasa reprezentująca osoby
- Osobę charakteryzują atrybuty: imię, nazwisko i wiek
- Zdefiniuj atrybuty jako właściwości i zarejestruj tylko gettery
- Utwórz kilka instancji tej klasy i wstaw je do listy
- Wypisz dane przechowywane w liście
- Wykorzystując funkcję klucza, posortuj osoby zgodnie z następującymi wytycznymi:
  - o kolejności decyduje nazwisko w porządku rosnącym
  - gdyby to nie było rozstrzygające, to w drugiej kolejności o porządku decyduje imię w porządku malejącym
- Zastanów się, jak tego dokonać
- Wypisz posortowane osoby, aby zweryfikować, czy sortowanie działa poprawnie
- Utwórz funkcję komparatora służącą do określenia relacji między dwoma osobami
- Funkcja komparatora:
  - jest funkcją dwuargumentową – przekazujemy jej dwie osoby, dla których określamy ich porządek
  - jeśli osoba pierwsza powinna poprzedzać osobę drugą (wg zadanych kryteriów sortowania), to należy zwrócić wartość -1
  - jeśli osoba druga powinna poprzedzać osobę pierwszą, to należy zwrócić wartość 1

  - jeśli kolejność jest nieistotna (stan używany do porównywania jest identyczny), to zwracamy 0

- Wykorzystując dekorator z modułu *functools*, skonwertuj komparator na funkcję klucza
- Posortuj listę osób wykorzystując, tak utworzoną funkcję klucza
- Porównaj wyniki obu sortowań

ĆWICZENIE 1.5:

Operatory relacyjne

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność:
  - przeciążania operatorów relacyjnych
  - automatycznego generowania metod definujących relacje

CELE I ZADANIA:

- Utwórz prostą klasę stanową i kilka jej instancji
- Celem ćwiczenia jest przeciążenie operatorów relacyjnych
- Nadpisz metody odpowiedzialne za działanie operatorów == oraz <
- Sprawdź, czy one działają poprawnie
- Zweryfikuj, co się stanie, gdy spróbujemy użyć operatorów >, <=, >=
- Wykorzystując odpowiedni dekorator z modułu *functools* wygeneruj brakujące metody odpowiedzialne za działanie powyższych operatorów
- Ponownie sprawdź, czy teraz operatory działają

ALGORYTM WYKONANIA:

- Utwórz prostą klasę zawierającą dane – możesz wykorzystać klasę z poprzedniego ćwiczenia
- Przedefinuj w niej metody magiczne odpowiedzialne za działanie operatorów == oraz < (jakie to metody?)
  - operator == powinien zwrócić wartość True, tylko wtedy, gdy nazwiska i imiona obu osób są identyczne
  - operator < powinien zwrócić wartość True, gdy nazwisko pierwszej osoby występuje w słowniku wcześniej niż nazwisko osoby drugiej, a w przypadku identycznych nazwisk – podobnie trzeba porównać imiona
- Utwórz kilka instancji osób i sprawdź działanie obu operatorów
- Zobacz także, czy działają pozostałe operatory relacyjne (>, <= oraz >=)
- Wykorzystując dekorator z modułu *functools*, dodaj do klasy brakujące metody odpowiedzialne za działanie powyższych operatorów
- Ponownie sprawdź, czy teraz operatory działają i czy działają poprawnie

ĆWICZENIE 1.6:

Redukcja

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność:
  - tworzenia nowych funkcji cząstkowych
  - wykorzystania funkcji redukcji

CELE I ZADANIA:

- Wykorzystaj funkcję redukcji z modułu *functools* do utworzenia nowych funkcji, które dla podanej sekwencji liczbowej zwrócą pojedynczą wartość

ALGORYTM WYKONANIA:

- Wykorzystaj funkcję redukcji z modułu *functools* do utworzenia nowych funkcji, które dla podanej sekwencji liczbowej zwrócą:
  - wartość minimalną
  - wartość maksymalną
  - sumę wartości
- Utwórz te funkcje jako funkcje cząstkowe (p. funkcja *partial*)
- Przetestuj działanie tych funkcji

ĆWICZENIE 1.7:

Przeciążanie funkcji

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność przeciążania funkcji

CELE I ZADANIA:

- Korzystając z możliwości modułu *functools* utwórz kilka wariantów funkcji o tej samej nazwie przyjmujących dane wejściowe różnych typów

ALGORYTM WYKONANIA:

- Utwórz funkcję o nazwie *week_day*, której zadaniem jest wypisanie nazwy dnia tygodnia dla podanej daty
- Datę wejściową można podać w postaci:
  - tekstu, zgodnie ze wzorcem *rok-miesiąc-dzień*
  - instancji typu *datetime.date*
  - słownika, z kluczami: *'year', 'month', 'day'*
- Do wypisania nazwy dnia tygodnia możesz użyć funkcji *strftime*
- Do przeciążenia funkcji użyj możliwości modułu *functools*
- Przetestuj działanie tych funkcji

2 KOLEKCJE

ĆWICZENIE 2.1:

Lotniska

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność grupowania danych:
  - w "klasyczny" sposób
  - z użyciem słownika *defaultdict*

CELE I ZADANIA:

- Pogrupuj dane wg zadanego kryterium na kilka kategorii
- Dokonaj tego z użyciem:
  - słownika typu *dict*
  - słownika typu *defaultdict*
- Porównaj zawartość obu słowników
- Porównaj czasy tworzenia obu słowników

ALGORYTM WYKONANIA:

- Jako dane do ćwiczenia możesz wykorzystać informacje o liczbie pasażerów obsłużonych przez wybrane lotniska w Polsce – znajdziesz je na końcu opisu tego ćwiczenia
- Każda z zagnieżdżonych krotek reprezentuje:
  - nazwę lotniska
  - rok z którego pochodzą dane
  - liczbę obsłużonych pasażerów
- Napisz funkcję, która te dane skonwertuje do słownika, gdzie:
  - kluczem będzie nazwa lotniska
  - zaś wartością słownik w którym:
    - kluczem jest rok
    - wartością – liczba pasażerów
- Napisz drugą funkcję, która zrobi to samo, ale wykorzysta słownik typu *defaultdict*
- Wywołaj obie funkcje i porównaj, czy oba słowniki mają taką samą zawartość
- Wykorzystując moduł *timeit* porównaj czasy wykonania obu metod
- Wykorzystując słownik *defaultdict*, napisz funkcję zwracającą zestawienie (słownik):
  - liczby pasażerów obsłużonych przez dane lotnisko (rok nieistotny)
  - liczby pasażerów obsłużonych w danym roku (lotnisko nieistotne)

DANE DO ĆWICZENIA:

```
airports = (
('Lotnisko Chopina w Warszawie', 2018, 17737231),
('Lotnisko Chopina w Warszawie', 2019, 18844591),
('Lotnisko Chopina w Warszawie', 2020, 5473224),
('Lotnisko Chopina w Warszawie', 2021, 7445468),

('Port Lotniczy Kraków-Balice', 2018, 6759683),
('Port Lotniczy Kraków-Balice', 2019, 8402859),
('Port Lotniczy Kraków-Balice', 2020, 2588970),
('Port Lotniczy Kraków-Balice', 2021, 3065957),

('Międzynarodowy Port Lotniczy Katowice-Pyrzowice', 2018, 4825845),
('Międzynarodowy Port Lotniczy Katowice-Pyrzowice', 2019, 4843650),
('Międzynarodowy Port Lotniczy Katowice-Pyrzowice', 2020, 1437876),
('Międzynarodowy Port Lotniczy Katowice-Pyrzowice', 2021, 2311586),

('Port Lotniczy Gdańsk-Rębiechowo', 2018, 4966949),
('Port Lotniczy Gdańsk-Rębiechowo', 2019, 5361134),
('Port Lotniczy Gdańsk-Rębiechowo', 2020, 1697406),
('Port Lotniczy Gdańsk-Rębiechowo', 2021, 2140522),

('Port Lotniczy Warszawa-Modlin', 2018, 3080699),
('Port Lotniczy Warszawa-Modlin', 2019, 3104277),
('Port Lotniczy Warszawa-Modlin', 2020, 870831),
('Port Lotniczy Warszawa-Modlin', 2021, 1455315),

('Port Lotniczy Wrocław-Strachowice', 2018, 3293948),
('Port Lotniczy Wrocław-Strachowice', 2019, 3543398),
('Port Lotniczy Wrocław-Strachowice', 2020, 1003066),
('Port Lotniczy Wrocław-Strachowice', 2021, 1409067),

('Port Lotniczy Poznań-Ławica', 2018, 2465418),
('Port Lotniczy Poznań-Ławica', 2019, 2372184),
('Port Lotniczy Poznań-Ławica', 2020, 652833),
('Port Lotniczy Poznań-Ławica', 2021, 1045751),

('Port Lotniczy Rzeszów-Jasionka', 2018, 769475),
('Port Lotniczy Rzeszów-Jasionka', 2019, 769252),
('Port Lotniczy Rzeszów-Jasionka', 2020, 234355),
('Port Lotniczy Rzeszów-Jasionka', 2021, 253210),

('Port Lotniczy Szczecin-Goleniów', 2018, 598663),
('Port Lotniczy Szczecin-Goleniów', 2019, 580479),
('Port Lotniczy Szczecin-Goleniów', 2020, 185848),
('Port Lotniczy Szczecin-Goleniów', 2021, 181849),

('Port Lotniczy Lublin', 2018, 454103),
('Port Lotniczy Lublin', 2019, 356011),
('Port Lotniczy Lublin', 2020, 123512),
('Port Lotniczy Lublin', 2021, 107423),
```

('Port Lotniczy Bydgoszcz', 2018, 398066),
('Port Lotniczy Bydgoszcz', 2019, 413472),
('Port Lotniczy Bydgoszcz', 2020, 124545),
('Port Lotniczy Bydgoszcz', 2021, 95118),

('Port Lotniczy Łódź', 2018, 217426),
('Port Lotniczy Łódź', 2019, 241707),
('Port Lotniczy Łódź', 2020, 75275),
('Port Lotniczy Łódź', 2021, 69296),

('Port Lotniczy Olsztyn-Mazury', 2018, 117102),
('Port Lotniczy Olsztyn-Mazury', 2019, 147446),
('Port Lotniczy Olsztyn-Mazury', 2020, 61114),
('Port Lotniczy Olsztyn-Mazury', 2021, 47039)
)

ĆWICZENIE 2.2:

Poliglota

UMIEJĘTNOŚCI:

- Po wykonaniu ćwiczenia zdobędziesz umiejętność wykorzystania kolekcji typu *Counter*

CELE I ZADANIA:

- Napisz program, który na podstawie dostarczonego tekstu spróbuje odgadnąć w jakim języku jest on napisany
- W algorytmie należy przeanalizować częstości występowania liter, które są specyficzne dla każdego języka
- Do zliczenia liter należy użyć klasy *Counter*

ALGORYTM WYKONANIA:

- Jako dane do ćwiczenia możesz wykorzystać średnie częstości występowania liter alfabetu dla kilku popularnych języków
- Możesz posłużyć się tabelą umieszczoną na stronie WWW pod adresem: https://en.wikipedia.org/wiki/Letter_frequency
- Napisz funkcję, która dla podanego pliku tekstowego zliczy ile w nim występuje różnych liter
  - pomiń wszystkie znaki, które nie są literami
  - zignoruj wielkość liter
  - do zliczenia liter wykorzystaj kolekcję typu *Counter*
- Napisz funkcję, która umożliwi znormalizowanie częstotliwości zliczonych liter (częstotliwości podane w Wikipedii są w procentach)
- Stosując metodę najmniejszych kwadratów znajdź najbardziej prawdopodobny język tekstu
  - należy obliczyć sumę kwadratów różnic częstości występowania każdej litery w języku i w tekście
  - najbardziej prawdopodobny będzie ten język, dla którego obliczona wartość będzie najmniejsza (czyli częstości występowania kolejnych liter są najbliższe)
- Przetestuj działanie programu dla kilku tekstów w różnych językach
- Pamiętaj, że im dłuższy tekst, tym bardziej trafne powinny być "przewidywania" programu