Programowanie aplikacji geoinformatycznych (Python)

Dla zadań, które mają odzwierciedlenie w konkretnym miejscu w kodzie, proszę o dodanie komentarza z numerem zadania.

GIT

1. Stwórz repozytorium i oprócz głównej gałęzi, stwórz na jej podstawie gałąź o nazwie **zajecia01**.
2. Na podstawie repozytorium zdalnego, stwórz połączone repozytorium lokalne.
3. Przejdź na utworzoną gałąź, dodaj tam folder **zajecia01** i wrzuć tam pliki z poprzednich zajęć.
4. Udostępnij zmiany z lokalnego repozytorium do zdalnego.
5. Stwórz pull request do głównej gałęzi i dodaj mnie jako recenzenta.

Wszystkie poniższe ćwiczenia proszę wykonać w podobnej konfiguracji, ale wykorzystując gałąź **zajecia02** oraz folder **zajecia02**. Gałąź proszę stworzyć na podstawie gałęzi głównej.

Zadanie obowiązkowe

Zaimportuj moduł **this** wykorzystując instrukcję **import this**.

Zapoznaj się z aforystycznymi zasadami, które powinny towarzyszyć każdemu programiście, który chce tworzyć lepszy i czytelniejszy kod w Pythonie.

Przypisania

6. Mając daną krotkę **dane = (2024, 'Python', 3.8)**, przypisz każdy element krotki do odpowiednich zmiennych: **rok**, **jezyk** i **wersja**. Wyświetl te zmienne.
7. Mając listę **oceny = [4, 3, 5, 2, 5, 4]**, przypisz pierwszą wartość do zmiennej **pierwsza**, ostatnią do **ostatnia**, a pozostałe do listy **srodek**. Wykorzystaj ***** do zgromadzenia środkowych wartości. Wyświetl te zmienne.
8. Dla krotki **info = ('Jan', 'Kowalski', 30, 'Polska', 'programista')**, przypisz imię do zmiennej **imie**, nazwisko do **nazwisko**, a zawód do **zawod**, ignorując pozostałe wartości. Do ignorowania wykorzystaj znak **_**. Wyświetl przypisane zmienne.
9. Mając zagnieżdżoną strukturę **dane = (2024, ['Python', 3.8, ('Stabilna', 'Wersja')])**, przypisz rok do zmiennej **rok**, nazwę języka do **jezyk**, wersję do **wersja** i opis wersji do zmiennej **opis**. Wyświetl te zmienne.

Przypisania z wieloma celami i współdzielone referencje

10. Stwórz zmienną **a** oraz **b**, użyj przypisania z wieloma celami i przypisz im listę **[1, 2, 3]**: **a = b = [1, 2, 3]**. Zmodyfikuj pierwszy element listy **b** przez przypisanie **b[0] = 'zmieniono'**. Wyświetl obie listy **a** i **b**, a następnie wyjaśnij, dlaczego zmiana w **b** wpłynęła również na **a**. Czy listy są obiektami mutowalnymi?
11. Korzystając z poprzedniego przykładu, utwórz zmienną **c** i przypisz jej kopię listy **a** (możesz użyć metody list() lub składni a[:]). Następnie zmodyfikuj pierwszy element w **c** i przypisz mu wartość **'nowa wartość'**. Wyświetl listy **a**, **b** i **c**, zauważając, że tym razem zmiana w **c** nie wpłynęła na **a** ani **b**. Wyjaśnij, dlaczego kopiowanie listy zapobiegło współdzieleniu referencji.
12. Utwórz zmienną **x** oraz **y**, przypisz im wartość 10 poprzez **x = y = 10**. Zwiększ wartość **y** o 1 (np. **y = y + 1**). Wyświetl wartości **x** i **y**, a następnie wyjaśnij, dlaczego modyfikacja **y** nie wpłynęła na wartość **x**. Czy integery są obiektami mutowalnymi?

Przypisania rozszerzone i współdzielone referencje

13. Wyzwól następujący kod, wyświetl **K**, **L**, **M** i **N**. Wyjaśnij w jaki sposób konkatenacja zachowuje się inaczej od przypisania rozszerzonego.

K = [1, 2]
L = K
K = K + [3, 4]
M = [1, 2]
N = M
M += [3, 4]

Techniki tworzenia pętli - uzupełnienie

14. Mając dwie listy, **imiona = ['Anna', 'Jan', 'Ewa']** i **oceny = [5, 4, 3]**, użyj **zip** do stworzenia pary każdego imienia z odpowiadającą mu oceną. Następnie, iteruj przez te pary, wyświetlając imię wraz z oceną. Co się stanie, jeśli listy będą miały różne długości?
15. Mając listę **liczby = [1, 2, 3, 4, 5]**, napisz funkcję **kwadrat(x)**, która zwraca kwadrat liczby x. Użyj **map** z tą funkcją, aby stworzyć nową listę, w której każdy element jest kwadratem odpowiadającego mu elementu z listy liczby. Wyświetl tą listę.

Iteratory

Dokumentowanie kodu

Funkcje

Argumenty

16. Napisz funkcję **zmien_wartosc(arg)**, która przyjmuje jeden argument i próbuje zmodyfikować ten argument w różny sposób w zależności od tego, czy jest on niemutowalny (w tym przypadku integerem) czy mutowalny (w tym przypadku listą).
    a. Jeśli jest listą, wykonaj **arg[0] = 'kalafior '**.
    b. Jeśli jest integerem, wykonaj **arg = 65482652**.

Przydatna może okazać się funkcja **isinstance()**.

Wypisz przykłady dla obu przypadków, wypisz wartości przed i po wykonaniu funkcji. Jak się zachowują te obiekty?

17. Napisz funkcję z**amowienie_produktu**, która przyjmuje jeden obowiązkowy argument pozycyjny **nazwa_produktu** i dwa obowiązkowe argumenty nazwane: **cena** i **ilosc**. Funkcja powinna zwracać tekst podsumowujący zamówienie, zawierające nazwę produktu, łączną cenę (cena * ilość) oraz ilość zamówionego produktu.
    a. Stwórz pustą listę, do której wstawisz wartości zwracane przez funkcję dla 3 różnych produktów.
    b. Przeiteruj po wypełnionej liście, wyświetl teksty.
    c. Zmodyfikuj funkcję tak, żeby oprócz tekstu podsumowującego zwracała także wartość zamówienia.
    d. Na koniec wyświetl sumaryczną wartość zamówień (sumę z każdego zamówionego produktu).
    e. Dodaj wartość domyślną dla argumentu **ilosc** równą 1.

Obowiązkowo wykorzystaj poniższy początek definicji funkcji:

```
def zamowienie_produktu(nazwa_produktu, *, cena, ilosc):
    pass
```

18. Napisz funkcję **stworz_raport**, która przyjmuje dowolną liczbę argumentów pozycyjnych (*args) i nazwanych (**kwargs). Argumenty pozycyjne powinny reprezentować numery ID produktów, a argumenty nazwane - informacje o tych produktach (np. nazwa, cena). Funkcja powinna tworzyć i wyświetlać raport, w którym dla każdego ID produktu podane są szczegółowe informacje na jego temat.

Wywołanie funkcji powinno wyglądać następująco:

```
stworz_raport(101, 102, 101_nazwa="Kubek termiczny", 101_cena="45.99 zł",
102_nazwa="Długopis", 102_cena="4.99 zł")
```

Zasięgi

Wprowadzone są na tym etapie, ale tak na prawdę dotyczą dostępności zmiennych, więc są także powiązane z przestrzeniami nazw.

Rozwiązywanie konfliktów w zasięgu nazw Pythona czasami nazywane jest regułą LEGB:

- L - Local (Lokalne),
- E - Enclosing (Lokalne funkcji otaczających),
- G - Global (Globalne) - w Pythonie globalny zasięg rozumiany jest na poziomie pojedynczego modułu (pliku),
- B - Built-in (Wbudowane).

Funkcje fabrykujące - obiekty funkcji, które pamiętają wartości w otaczających zasięgach

19. Napisz funkcję fabrykującą **stworz_funkcje_potegujaca(wykladnik)**, która przyjmuje jeden argument: wykładnik potęgi. Zwracana przez nią funkcja zagnieżdżona **poteguj(podstawa)** powinna również przyjmować jeden argument – podstawę potęgi – i zwracać wynik podniesienia tej podstawy do potęgi określonej przez wykładnik przekazany do funkcji zewnętrznej.

Wywołanie takiej funkcji i sprawdzenie powinno wyglądać następująco:

```
potega_2 = stworz_funkcje_potegujaca(2)  # Tworzy funkcję potęgującą do kwadratu
print(potega_2(4))  # Wynik: 16
```

20. Napisz funkcję **licznik()**, która za każdym razem, gdy jest wywoływana, zwiększa swoje wewnętrzne liczenie o jeden (licznik stanu). Zaimplementuj cztery wersje tej funkcji, wykorzystując:
    a. Zmienną nonlocal w zagnieżdżonej funkcji.
    b. Zmienną global.
    c. Klasę z atrybutem instancji. # Wskazówka: zaimplementowanie w klasie funkcji __init__ oraz __call__
    d. Atrybut funkcji. # Funkcja, jak każdy inny obiekt, może mieć swoje atrybuty

Adnotacje

21. Dodaj adnotacje typów argumentów oraz wartości zwracanej do wybranej przez siebie funkcji.

Funkcje anonimowe – lambda

22. Masz daną listę słowników reprezentujących informacje o książkach w bibliotece. Każdy słownik zawiera klucze: **'tytul'**, **'autor'** oraz **'rok_wydania'**. Twoim zadaniem jest napisanie kodu, który wykonuje następujące operacje przy użyciu funkcji **lambda**:
    a. Sortowanie książek według roku wydania: Posortuj listę książek w kolejności rosnącej według roku ich wydania.
    b. Filtracja książek wydanych po 2000 roku: Utwórz nową listę zawierającą tylko te książki, które zostały wydane po roku 2000.
    c. Transformacja listy do listy tytułów: Przekształć oryginalną listę książek w listę zawierającą tylko tytuły książek.

Wykorzystaj funkcje sorted(), filter() oraz map() w połączeniu z funkcjami lambda do realizacji zadania.

Generatory – funkcje generatorów i wyrażenia generatorów

23. Napisz generator, który iteracyjnie zwraca nazwy dni tygodnia: od poniedziałku do niedzieli. Następnie, użyj tego generatora w pętli, aby wyświetlić każdy dzień tygodnia. Dodatkowo, zademonstruj, jak można użyć tego generatora do pobrania tylko pierwszych trzech dni tygodnia bez konieczności iterowania przez cały tydzień.

To zadanie można wykonać zarówno funkcją jak i wyrażeniem.

Pakiety modułów

Jako podstawową ścieżkę odniesienia do podziału na pakiety proszę uznać folder **zajecia02** w repozytorium.

24. Rozbij całość napisanego do tej pory kodu na jeden główny skrypt w folderze ./scripts/run.py (który odpowiedzialny będzie za samo wywołanie funkcji) oraz na pakiet (stworzony w folderze ./zajecia02), który będzie zawierał osobne moduły oraz subpakiet (stworzony w folderze ./zajecia02/liczniki_stanu) w którym w osobnych modułach zapisane będą różne wersje funkcji z zadania 20. Pakiety i subpakiety mają wykorzystywać listę __all__.