1. Własne klasy, tworzenie instancji obiektów, konstruktor/inicjalizator, metody i atrybuty instancyjne, atrybuty i metody klasowe

1. **Co to jest klasa w Pythonie?**

   Klasa to definicja struktury obiektu, czyli "przepis", który mówi, jakie właściwości i funkcje (metody) mają obiekty

2. **Jak utworzyć instancję obiektu klasy?**

   Instancję obiektu klasy tworzymy, wywołując klasę jak funkcję, np. obiekt = Klasa().

3. **Czym jest konstruktor (__init__()) i do czego służy?**

   Konstruktor __init__() to metoda specjalna, która uruchamia się przy tworzeniu obiektu i inicjalizuje jego atrybuty.

4. **Co to jest atrybut instancyjny? Jak się go definiuje**?

   Atrybut instancyjny to cecha obiektu przypisana do konkretnej instancji, np. self.imie = "Jan".

5. **Co to jest metoda instancyjna? Jak się ją wywołuje?**

   Metoda instancyjna to funkcja wewnątrz klasy, która operuje na konkretnej instancji i zawsze ma pierwszy parametr self.

6. **Co to jest atrybut klasowy i czym różni się od atrybutu instancyjnego?**

   Atrybut klasowy jest wspólny dla wszystkich obiektów i jest definiowany w ciele klasy.

7. **Jak zdefiniować metodę klasową i czym różni się od metody instancyjnej?**

   Metoda klasowa ma dekorator @classmethod i jako pierwszy parametr przyjmuje cls (klasę zamiast instancji).

---

2. Pojęcie obiektu, sposoby reprezentacji obiektów, obiekt iterowalny i iterator

1. **Co to jest obiekt w Pythonie?**

   Obiekt to konkretny egzemplarz klasy, który ma swoje dane i funkcje.

2. **Jakie metody służą do reprezentacji obiektów w formie tekstowej (__str__(), __repr__()) i czym się różnią?**

   __str__() zwraca czytelną reprezentację obiektu (np. dla użytkownika), __repr__() zwraca szczegółową, bardziej techniczną reprezentację (dla programisty).

3. **Co to jest obiekt iterowalny? Podaj przykład iterowalnego obiektu.**

   Obiekt iterowalny to obiekt, który można "przechodzić" w pętli for, np. lista, tupla, string.

4. **Co to jest iterator? Jakie metody musi posiadać, aby był iteratorem?**

Iterator to obiekt, który ma metody __iter__() i __next__() do zwracania kolejnych elementów.

5. **Jaka jest różnica między iter() a next()?**

   iter() zwraca iterator z obiektu iterowalnego, next() zwraca kolejny element iteratora.

---

3. Właściwości (property) i przeciążanie operatorów

1. **Co to jest właściwość (property) w Pythonie?**

   property pozwala zamienić metodę na coś, co wygląda jak zwykły atrybut.

2. **Jak utworzyć getter i setter za pomocą @property?**

   Getter tworzymy za pomocą @property, a setter za pomocą @nazwa.setter.

3. **Jak wygląda dostęp do właściwości w obiekcie (jak pobierasz i ustawiasz wartość)?**

   Dostęp do właściwości odbywa się poprzez obiekt.wlasciwosc (bez nawiasów).

4. **Co to jest przeciążenie operatora?**

   Przeciążenie operatora to nadpisanie zachowania operatora, np. +, -, ==, aby działał dla własnych obiektów.

5. **Jak przeciążyć operator + w Pythonie? Jaką metodę należy nadpisać?**

   Aby przeciążyć operator +, nadpisujemy metodę __add__().

---

4. Dziedziczenie, agregacja/kompozycja, super(), protokoły i klasy abstrakcyjne

1. **Co to jest dziedziczenie i jak je zaimplementować w Pythonie?**

   Dziedziczenie to mechanizm umożliwiający tworzenie nowych klas na podstawie istniejących (class NowaKlasa(StaraKlasa):).

2. **Co robi funkcja super()?**

   super() pozwala wywołać metodę z klasy bazowej.

3. **Czym różni się agregacja od kompozycji? Podaj przykład dla obu.**

   gregacja to sytuacja, gdy obiekt zawiera inny obiekt, ale go nie tworzy, np. samochód z istniejącym silnikiem. **Kompozycja** to sytuacja, gdy obiekt tworzy inny obiekt wewnętrznie i go kontroluje, np. samochód, który tworzy swoje koła.

4. **Co to jest klasa abstrakcyjna i jak się ją definiuje w Pythonie?**

   Klasa abstrakcyjna to klasa, której nie można instancjonować bezpośrednio i która może mieć abstrakcyjne metody (@abstractmethod).

5. **Czym jest protokół w Pythonie i jakie jest jego zastosowanie?**

   rotokół definiuje wymagania dla obiektu (np. __iter__(), __next__()), bez narzucania dziedziczenia.

---

5. Paradygmat funkcyjny w Pythonie

1. **Jak w Pythonie funkcje mogą być traktowane jako obiekty?**

   W Pythonie funkcje są obiektami – można je przypisać do zmiennej lub przekazać jako argument.

2. **Co to są funkcje wyższego rzędu? Podaj przykłady takich funkcji w Pythonie.**

   Funkcje wyższego rzędu to funkcje, które przyjmują inne funkcje jako argumenty lub zwracają funkcje (map(), filter()).

3. **Czym jest funkcja anonimowa lambda i jak ją utworzyć?**

   lambda to funkcja anonimowa, np. lambda x, y: x + y.

4. **Do czego służy funkcja map()? Podaj przykład.**

   map() stosuje funkcję do każdego elementu iterowalnego obiektu.

5. **Jak działa funkcja filter()?**

   filter() zwraca elementy, które spełniają warunek określony funkcją.

6. **Co robi reduce() i z jakiej biblioteki pochodzi?**

   reduce() składa wszystkie elementy iterowalnego obiektu do jednej wartości i pochodzi z functools.

7. **Co to jest dekorator i jak działa?**

Dekorator to specjalna funkcja, która **modyfikuje działanie innej funkcji lub metody** bez zmieniania jej kodu źródłowego. Dekorator "opakowuje" funkcję w dodatkowe instrukcje – **coś robi przed wywołaniem funkcji i/lub po jej wywołaniu**. Dzięki temu możesz łatwo dodać nowe funkcjonalności, np. logowanie, sprawdzanie uprawnień, mierzenie czasu wykonania itp.

---

6. Deskryptory i metaklasy

1. **Co to jest deskryptor i jakie metody musi implementować?**

   Deskryptor to obiekt kontrolujący dostęp do atrybutu klasy za pomocą metod __get__(), __set__(), __delete__().

2. **Podaj przykład deskryptora z metodą __get__() i __set__().**

```
1. class Deskryptor:
2.     def __get__(self, instance, owner):
```

```
3.         return instance._wartosc
4.
5.     def __set__(self, instance, value):
6.         instance._wartosc = value
7.
```

3. Co to jest metaklasa i jaka jest domyślna metaklasa w Pythonie?

Metaklasa to "klasa dla klas", która kontroluje sposób ich tworzenia. Domyślną metaklasą w Pythonie jest type.

4. Jak można zdefiniować własną metaklasę?

Własną metaklasę definiujemy jako klasę dziedziczącą po type, np. class MyMeta(type):.

5. W jakich sytuacjach używa się metaklas?

Metaklasy są używane, gdy chcemy kontrolować, jak klasy są tworzone lub modyfikować ich strukturę.