

Python: Wzorce projektowe

Jerzy Grynczewski

Plan na dzisiaj

- ❖ Co to są wzorce projektowe ?

Plan na dzisiaj

- ❖ Co to są wzorce projektowe ?
- ❖ Czemu ich potrzebujemy ?

Plan na dzisiaj

- ❖ Co to są wzorce projektowe ?
- ❖ Czemu ich potrzebujemy ?
- ❖ Klasyfikacja wzorców projektowych.

Plan na dzisiaj

- ❖ Co to są wzorce projektowe ?
- ❖ Czemu ich potrzebujemy ?
- ❖ Klasyfikacja wzorców projektowych.
- ❖ Zasady programowania obiektowego (SOLID)

Plan na dzisiaj

- ❖ Co to są wzorce projektowe ?
- ❖ Czemu ich potrzebujemy ?
- ❖ Klasyfikacja wzorców projektowych.
- ❖ Zasady programowania obiektowego (SOLID)
- ❖ Interfejsy w Pythonie

Plan na dzisiaj

- ❖ Co to są wzorce projektowe ?
- ❖ Czemu ich potrzebujemy ?
- ❖ Klasyfikacja wzorców projektowych.
- ❖ Zasady programowania obiektowego (SOLID)
- ❖ Interfejsy w Pythonie
- ❖ Wzorzec - Strategia

Plan na dzisiaj

- ❖ Co to są wzorce projektowe ?
- ❖ Czemu ich potrzebujemy ?
- ❖ Klasyfikacja wzorców projektowych.
- ❖ Zasady programowania obiektowego (SOLID)
- ❖ Interfejsy w Pythonie
- ❖ Wzorzec – Strategia
- ❖ Wzorzec - Obserwator

Co to wzorzec projektowy

Wzorzec projektowy to modelowe rozwiązanie powszechnie występującego problemu projektowego. Opisuje problem i ogólne podejście do jego rozwiązania.

Co to wzorzec projektowy

Wzorzec projektowy to modelowe rozwiązanie powszechnie występującego problemu projektowego. Opisuje problem i ogólne podejście do jego rozwiązania.

Each pattern describes a problem which occurs over and over again in our environment and then describes the core of the solution to the problem".

Christopher Alexander (architect and design theorist not a software developer), a Pattern Language, Oxford University Press, New York, 1977.

Przykłady wzorców projektowych

Architektura budynków

Przykłady wzorców projektowych

Architektura budynków

Kody elektroniczne i
hydrauliczne

Przykłady wzorców projektowych

Architektura budynków

Kody elektroniczne i
hydrauliczne

Samochody

Przykłady wzorców projektowych

Architektura budynków

Kody elektroniczne i
hydrauliczne

Samochody

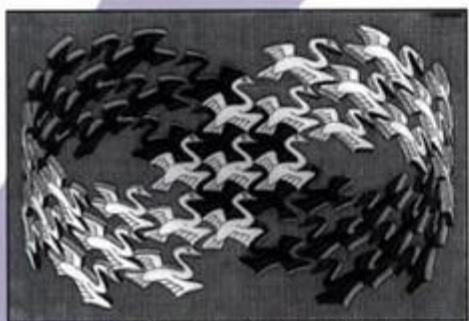
Interfejsy telefonów
komórkowych

Wzorce projektowe dają nam pewność, że wyniki naszej pracy są spójne, niezawodne i zrozumiałe.

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Pierwszy raz opublikowana w 1995

Gamma, Helm, Johnson, Vlissides – Gang of Four

Pierwsza kompletna publikacja dotycząca wzorców projektowych.

Do tej pory pozostaje niepodważalnym autorytetem.

Podstawowa klasyfikacja wzorców projektowych

Kreacyjne

Tworzenie obiektów

Podstawowa klasyfikacja wzorców projektowych

Kreacyjne

Tworzenie obiektów

Strukturalne

Kompozycja obiektów

Podstawowa klasyfikacja wzorców projektowych

Kreacyjne

Tworzenie obiektów

Strukturalne

Kompozycja obiektów

Behawioralne (Czynnościowe)

Interakcja pomiędzy obiektami i
odpowiedzialność

Alternatywna klasyfikacja wzorców projektowych

Klasowe

Opisujące statyczne związki
pomiędzy klasami

Alternatywna klasyfikacja wzorców projektowych

Klasowe

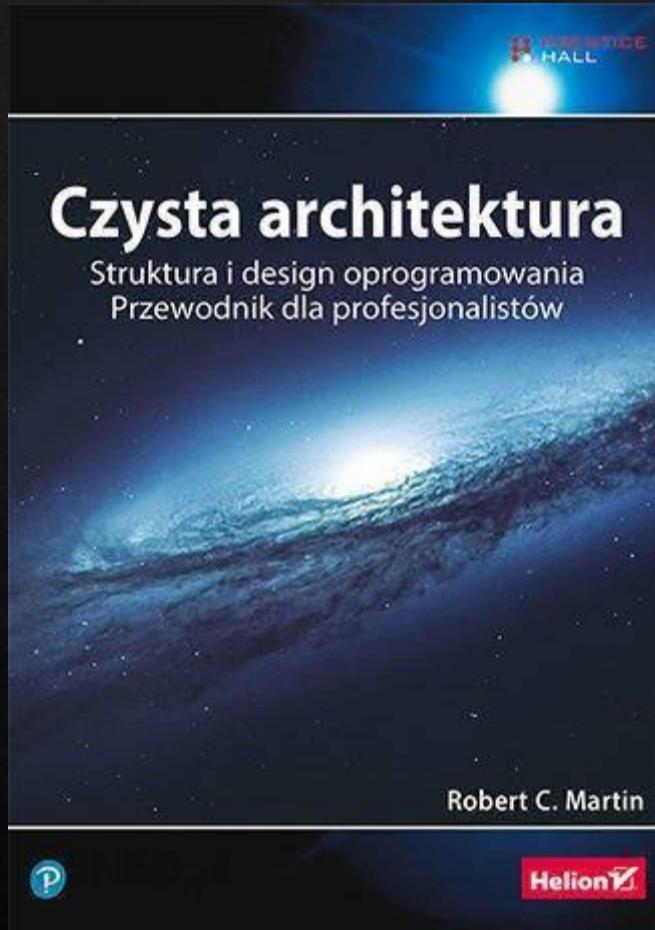
Opisujące statyczne związki
pomiędzy klasami

Obiektowe

Opisujące dynamiczne związki
pomiędzy obiektami

SOLID Principles of Object Oriented Design

SOLID Principles of Object Oriented Design



Robert C. Martin (wujek Bob)

SOLID Principles of Object Oriented Design

S

Reguła jednej
odpowiedzialności

SRP (Single Responsibility Principle)

S – Single Responsibility Principle



Każdy moduł powinien mieć jedną i tylko jedną przyczynę zmiany.

Klasa powinna mieć tylko jedną odpowiedzialność, czyli albo gotowanie, albo zmywanie, ale nigdy to i to.

SOLID Principles of Object Oriented Design

S

Reguła jednej
odpowiedzialności

SRP (Single Responsibility Principle)

O

Reguła otwarте-zamknięte

OCP (Open-Close Principle)

O - Open Close Principle



Element oprogramowania powinien być otwarty na rozbudowę, a zamknięty na modyfikacje.

Klasa powinna być otwarta na rozbudowę (przeważnie poprzez zastosowanie dziedziczenia), ale zamknięta na modyfikacje.

SOLID Principles of Object Oriented Design

S

Reguła jednej
odpowiedzialności

SRP (Single Responsibility Principle)

O

Reguła otwarте-zamknięte

OCP (Open-Close Principle)

L

Zasada podstawień Barbary
Liskov

LSP (Liskov Substitution Principle)

L – Liskov Substitution Principle



Podklasy powinny być w stanie zastąpić swoich rodziców bez wywoływania błędu w programie.

SOLID Principles of Object Oriented Design

S

Reguła jednej odpowiedzialności

SRP (Single Responsibility Principle)

O

Reguła otwarте-zamknięte

OCP (Open-Close Principle)

L

Zasada podstawień Barbary Liskov

LSP (Liskov Substitution Principle)

I

Zasada rozdzielania interfejsów

ISP (Interface Segregation Principle)

I – Interface Segregation Principle



Wiele różnych interfejsów jest lepsze niż jeden interfejs typu do-it-all

SOLID Principles of Object Oriented Design

S

Reguła jednej odpowiedzialności

SRP (Single Responsibility Principle)

O

Reguła otwarте-zamknięte

OCP (Open-Close Principle)

L

Zasada podstawień Barbary Liskov

LSP (Liskov Substitution Principle)

I

Zasada rozdzielenia interfejsów

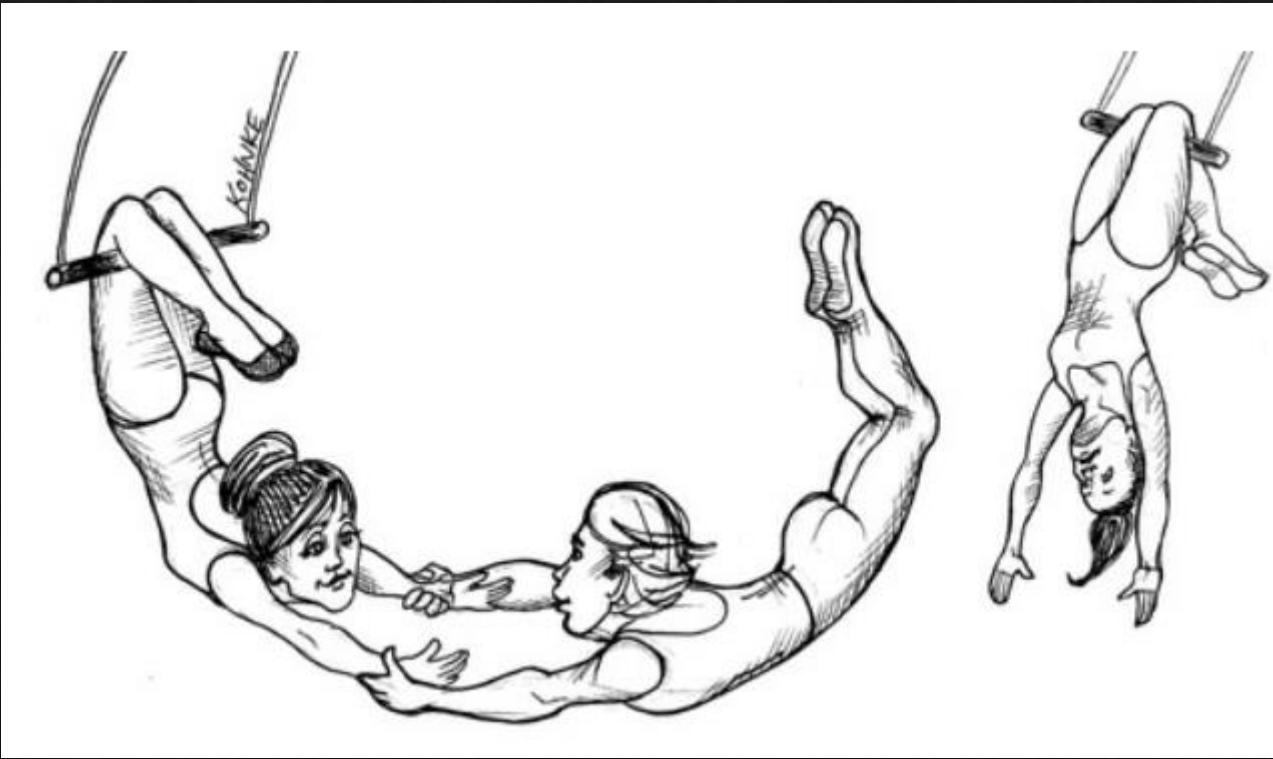
ISP (Interface Segregation Principle)

D

Zasada odwrócenia zależności

DIP (Dependency Inversion Principle)

D – Dependency Inversion Principle



Powinniśmy programować za pomocą abstrakcji, nie implementacji. Implementacje mogą się zmieniać, abstrakcje nie powinny

SOLID Principles of Object Oriented Design

S

Reguła jednej odpowiedzialności

SRP (Single Responsibility Principle)

O

Reguła otwarte-zamknięte

OCP (Open-Close Principle)

L

Zasada podstawień Barbary Liskov

LSP (Liskov Substitution Principle)

I

Zasada rozdzielenia interfejsów

ISP (Interface Segregation Principle)

D

Zasada odwrócenia zależności

DIP (Dependency Inversion Principle)

Interfejsy w Pythonie

I w SOLID

Interfejsy w Pythonie

I w SOLID

Wsparcie w Java,
C#, Visual Basic
dla definicji
interfejsu

Interfejsy w Pythonie

I w SOLID

Wsparcie w Java,
C#, Visual Basic
dla definicji
interfejsu

Wsparcie w C++
za pomocą klas
abstrakcyjnych

Interfejsy w Pythonie

I w SOLID

Wsparcie w Java,
C#, Visual Basic
dla definicji
interfejsu

Wsparcie w C++
za pomocą klas
abstrakcyjnych

Domyślnie brak
wsparcia w
Pythonie

Interfejsy w Pythonie

I w SOLID

Wsparcie w Java,
C#, Visual Basic
dla definicji
interfejsu

Wsparcie w C++
za pomocą klas
abstrakcyjnych

Domyślnie brak
wsparcia w
Pythonie

Wprowadzone
przez PEP 3119
za pomocą klas
abstrakcyjnych

Interfejsy w Pythonie

I w SOLID

Wsparcie w Java,
C#, Visual Basic
dla definicji
interfejsu

Wsparcie w C++
za pomocą klas
abstrakcyjnych

Domyślnie brak
wsparcia w
Pythonie

Wprowadzone
przez PEP 3119
za pomocą klas
abstrakcyjnych

Pierwszy raz
pojawiły się w
Pythonie 2.6 i 3

Definiowanie abstrakcyjnych klas bazowych

Moduł abc

```
import abc

class MyABC(metaclass=abc.ABCMeta):
    """Abstract Base Class Definition"""

    @abc.abstractmethod
    def do_something(self, value):
        """Required method"""

    @property
    @abc.abstractmethod
    def some_property(self):
        """Required property"""
```

Definiowanie abstrakcyjnych klas bazowych

Moduł abc

```
import abc
```

Klasa abstrakcyjna

```
class MyABC(metaclass=abc.ABCMeta):
    """Abstract Base Class Definition"""

    @abc.abstractmethod
    def do_something(self, value):
        """Required method"""

    @property
    @abc.abstractmethod
    def some_property(self):
        """Required property"""
```

Definiowanie abstrakcyjnych klas bazowych

Moduł abc

```
import abc
```

Klasa abstrakcyjna

```
class MyABC(metaclass=abc.ABCMeta):  
    """Abstract Base Class Definition"""
```

Metoda
abstrakcyjna

```
@abc.abstractmethod  
def do_something(self, value):  
    """Required method"""
```

```
@property  
@abc.abstractmethod  
def some_property(self):  
    """Required property"""
```

Definiowanie abstrakcyjnych klas bazowych

Moduł abc

```
import abc
```

Klasa abstrakcyjna

```
class MyABC(metaclass=abc.ABCMeta):  
    """Abstract Base Class Definition"""
```

Metoda
abstrakcyjna

```
@abc.abstractmethod  
def do_something(self, value):  
    """Required method"""
```

Property
abstrakcyjne

```
@property  
@abc.abstractmethod  
def some_property(self):  
    """Required property"""
```

Implementacja konkretnej klasy

Dziedziczy z ABC

```
class MyClass(MyABC):
    """Implementation of MyABC"""

    def __init__(self, value=None):
        self._my_prop = value

    def do_something(self, value):
        """Implementation of abstract method"""
        self._myprop *= 2+value

    @property
    def some_property(self):
        """Implementation of abstract property"""
        return self._myprop
```

Implementacja konkretnej klasy

Dziedziczy z ABC

```
class MyClass(MyABC):  
    """Implementation of MyABC"""
```

Standardowy konstruktor

```
def __init__(self, value=None):  
    self._my_prop = value
```

```
def do_something(self, value):  
    """Implementation of abstract method"""  
    self._myprop *= 2+value
```

```
@property  
def some_property(self):  
    """Implementation of abstract property"""  
    return self._myprop
```

Implementacja konkretnej klasy

Dziedziczy z ABC

```
class MyClass(MyABC):  
    """Implementation of MyABC"""
```

Standardowy konstruktor

```
def __init__(self, value=None):  
    self._my_prop = value
```

Implementacja abstrakcyjnej metody

```
def do_something(self, value):  
    """Implementation of abstract method"""  
    self._myprop *= 2+value
```

```
@property  
def some_property(self):  
    """Implementation of abstract property"""  
    return self._myprop
```

Implementacja konkretnej klasy

Dziedziczy z ABC

```
class MyClass(MyABC):  
    """Implementation of MyABC"""
```

Standardowy konstruktor

```
def __init__(self, value=None):  
    self._my_prop = value
```

Implementacja abstrakcyjnej metody

```
def do_something(self, value):  
    """Implementation of abstract method"""  
    self._myprop *= 2+value
```

Implementacja abstrakcyjnego property

```
@property  
def some_property(self):  
    """Implementation of abstract property"""  
    return self._myprop
```

A co jeżeli nie zaimplementujemy abstrakcyjnej metody ?

A co jeżeli nie zaimplementujemy abstrakcyjnej metody ?

TypeError: Can't instantiate abstract class BadClass with abstract methods
do_something, some_property

Wzorce projektowe

1. Strategia (Strategy aka Policy pattern)



Complexity: ★★★

Popularity: ★★★★

1. Strategia (Strategy)



Complexity: ★★★

Popularity: ★★★★

Czynnościowy wzorzec projektowy, który zamienia zbiór czynności (rodzinę wymiennych algorytmów) w obiekty i czynni je zmiennymi wewnętrznymi korzystającego z nich obiektu.

1. Strategia (Strategy)

Motywacja:

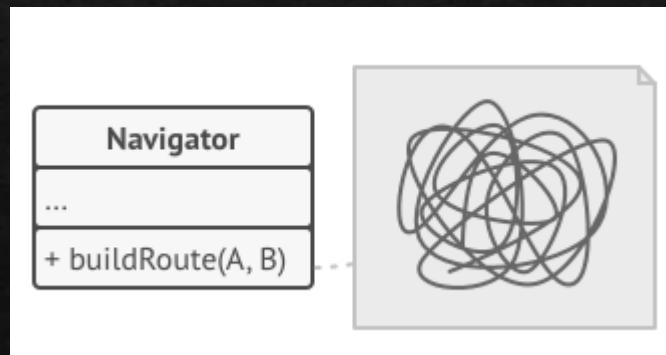
Tworzymy aplikację wyświetlającą trasę na mapie.

1. Strategia (Strategy)

Motywacja:

Tworzymy aplikację wyświetlającą trasę na mapie.

Problem:

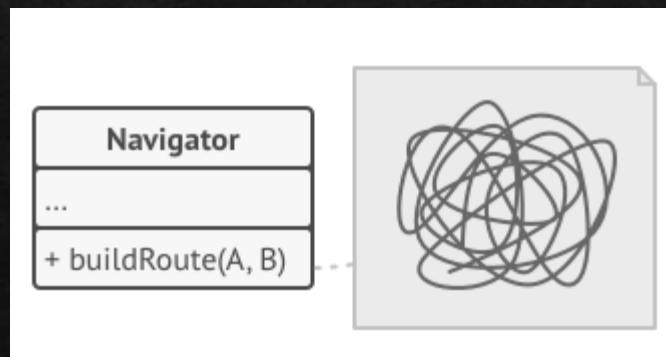


1. Strategia (Strategy)

Motywacja:

Tworzymy aplikację wyświetlającą trasę na mapie.

Problem:

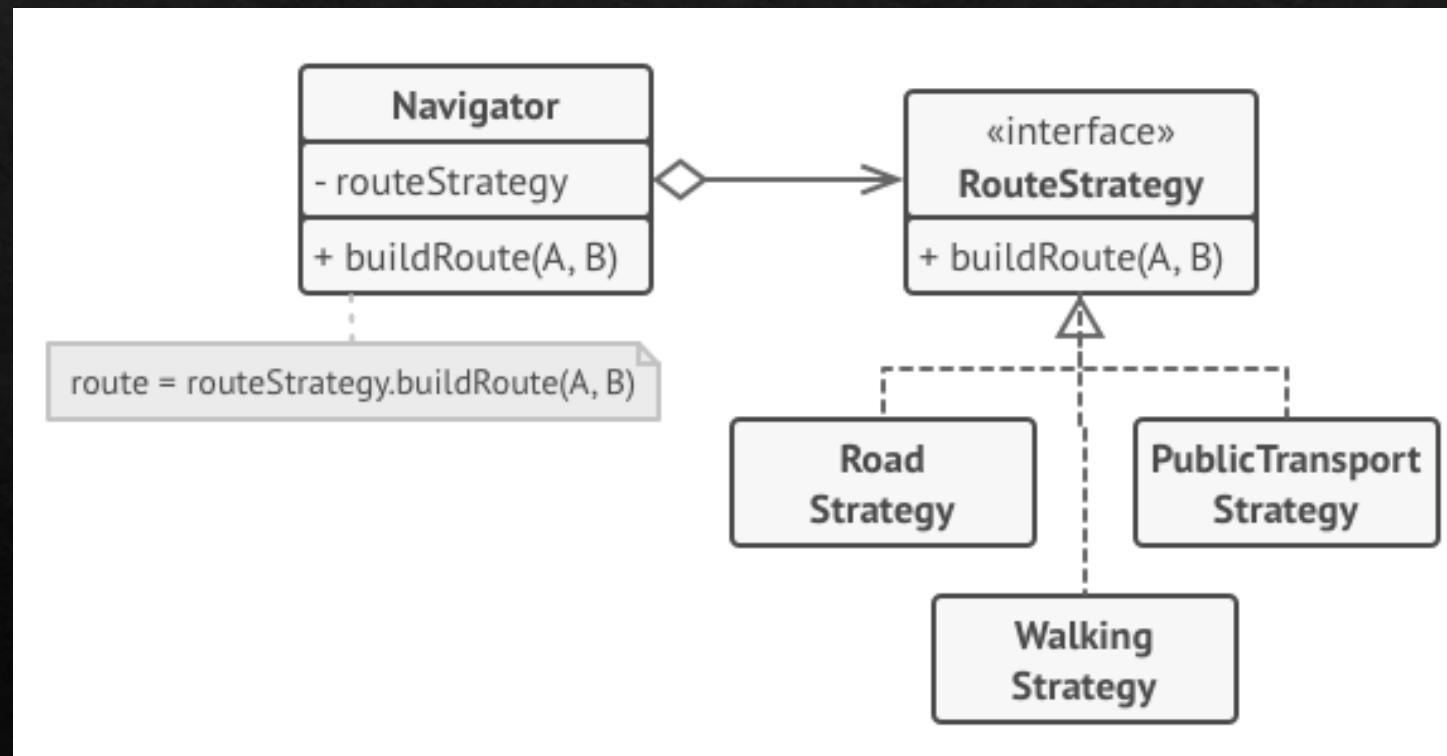


Z biznesowego punktu widzenia aplikacja może być sukcesem, ale techniczne aspekty przyprawiają cię o coraz większy ból głowy. Za każdym razem kiedy dodajesz nowy algorytm do liczenia trasy, główna klasa Navigator podwaja swoją wielkość. W którymś momencie staje się zbyt duża dla utrzymania.

1. Strategia (Strategy)

Rozwiązanie:

Strategia definiuje rodzinę algorytmów (tutaj wytyczania trasy), kapsułkuje każdy z nich w oddzielnej klasie i zapewnienie wymienność powstałych obiektów poprzez zastosowanie interfejsu.



1. Strategia (Strategy)

Przykład praktyczny - Kalkulator kosztów przesyłki

Specyfikacja.

Musi uwzględniać:

- Federal Express (FedEx)
- UPS
- Poczta Polska

Powinien zapewniać łatwą rozbudowę o nowe firmy kurierskie.

1. Strategia (Strategy)

Napotkane problemy

Złamanie reguły pojedynczej odpowiedzialności (S)

Złamanie reguły otwarte/zamknięte (O)

Złamanie reguły odwróconych zależności (D)

Długa lista if/elif/else

Naprawmy je

1. Strategia (Strategy)

Co udało się osiągnąć?

Uniknąć łamania reguł SOLID

Umożliwić testowanie algorytmów w izolacji

Umożliwić testowanie kodu zewnętrznego za pomocą moków

Nie ma już wydłużającej się listy if/elif/else

Naprawmy je