Programowanie obiektowe

Wykład 1

Piotr Błaszyński

1 kwietnia 2022

Zagadnienia

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Uwaga: dużo trudnych słów - wszystkie (w swoim czasie) zostaną omówione):

- OOP - Programowanie obiektowe
- Klasy
  - Enkapsulacja
  - Hermetyzacja
- Obiekty
- Dziedziczenie
  - Polimorfizm
  - Dziedziczenie wielobazowe
  - Interfejsy
- Wyjątki
- Szablony, abstrakcyjne typy danych a duck typing w pythonie

Architektura

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Architektura oprogramowania to zbiór zasad i technik:

- Dzielenie problemów i budowanie systemu z małych kawałków
- Techniki stosowane do tworzenia połączeń (interfejsów) pomiędzy tymi kawałkami
- Techniki wykorzystywane do zarządzania strukturą i przepływem
- Techniki stosowane do połączenia (interfejsu) systemu ze środowiskiem zewnętrznym
- Właściwe korzystanie z podejść do wytwarzania, rozwoju i dostarczania oprogramowania, technik i narzędzi.

Architektura - czemu jest ważna?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Podstawowym celem architektury oprogramowania jest określenie wymagań niefunkcjonalnych systemu i określenie środowiska jego pracy. Określa to jak dostarczyć funkcjonalność w ramach ustalonych zasad architektonicznych. Architektura jest ważna, ponieważ:

- Pozwala kontrolować złożoność
- Wymusza najlepsze praktyki
- Zapewnia spójność i jednolitość systemu
- Zwiększa przewidywalność
- Pozwala na ponowne użycie.

OOP - programowanie obiektowe

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Podstawowe pojęcia

- Object oriented programming - OOP
- Programowanie zorientowane obiektowo - mniej popularne tłumaczenie,
- Programowanie obiektowe
- Object oriented design - sposób planowania, później również implementacji oprogramowania w taki sposób, że dzielimy problem z myślą o obiektach (duże uproszczenie - pozostańmy jednak przy takiej definicji)

OOP - co to jest?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Wszystko w OOP jest grupowane jako samowystarczalne obiekty. Dzięki temu zyskujemy możliwość ponownego użycia wcześniej napisanego kodu (pamiętamy o zasadzie Don't Repeat Yourself tak mocno akcentowanej w Pythonie) za pomocą czterech głównych koncepcji programowania obiektowego (na razie tylko lista):

1. Enkapsulacja (ang. Encapsulation)
2. Abstrakcja (ang. Abstraction)
3. Dziedziczenie (ang. Inheritance)
4. Polymorfizm (Polymorphism).

Pierwszy obiekt

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Dla lepszego zrozumienia modelu zorientowanego obiektowo, weźmy swoją "rękę" jako przykład. "Dłoń" (lub "Ręka") jest klasą. Nasze (u większości) ciało składa się z dwóch obiektów typu "ręka", o nazwach "lewa ręka" i "prawa ręka".

Pierwszy obiekt

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Ich główne funkcje są kontrolowane lub zarządzane przez zbiór sygnałów elektrycznych przesyłanych za pośrednictwem ramion (czyli przez pewien interfejs),
- Więc ramię jest interfejsem, który organizm wykorzystuje do interakcji z rękoma,
- Ręka jest klasą o dobrej architekturze.
- Ręka jest ponownie wykorzystywana do tworzenia lewej ręki i prawej ręki,
  - oczywiście trzeba lekko zmienić właściwości tego konkretnego obiektu.

Pierwsze pytanie

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Jakie jeszcze obiekty mamy w tej sali/pomieszczeniu?
- Czy można je policzyć?
- Jakie cechy maja te obiekty?
- Czy identyczne krzesła to też obiekty?

Więc? Co to jest obiekt?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Pewna rzecz która może wykonywać określone czynności. Zbiór tych czynności definiuje zachowania obiektów,
- Przykładowo: prawa ręka (jako obiekt) może chwytać, noga (jako obiekt) może kopać, stać, poruszać się, Obiekt reprezentujący studenta może nam zaprezentować jego imię i nazwisko albo numer indeksu,
- Z założenia można odróżnić od siebie dwa obiekty,

Co to jest obiekt? Bliżej kodu.

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- W rozumieniu programowania obiektowego obiekt jest instancją klasy
- W takim razie czym jest klasa?

Klasa

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Klasa jest po prostu przedstawieniem typu obiektu.
- Jest to opis, plan, definicja lub jeszcze inaczej szablon, który opisuje szczegóły obiektu.
- Klasa jest wzorcem, z którego tworzone są poszczególne obiekty.
- Klasa składa się z trzech rzeczy: nazwa, atrybuty (właściwości) i operacje.

Klasa - pytania

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Jak dokładnie opisać obiekty (jakie klasy) jakie mamy w tej sali/pomieszczeniu?
  - Klasa składa się z trzech rzeczy: nazwa, atrybuty (właściwości) i operacje.
- Jaka klasa jest najliczniejsza, ma najwięcej obiektów?
- Czy zawsze wszystko modelować, czy opisywać wszystkie cechy?
  - To bardziej pytanie do prowadzącego. W skrócie - NIE.
- Co z tymi krzesłami, jak je odróżniać?

Klasy - Nie było nic o pythonie?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Było: DRY

Python jest oczywiście językiem wspierającym programowanie obiektowe (górnolotnie mówiąc: wspiera paradygmat programowania obiektowego). Dostarcza większość popularnych w innych językach mechanizmów programowania obiektowego.

Jak utworzyć klasę w pythonie?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Od razu utworzymy i klasę i z niej skorzystamy, na kolejnych slajdach omówienie poszczególnych konstrukcji, również uproszczenie kodu:

```
class Reka:
    def start(self, ktora):
        self.ktora=ktora
    def pokaz(self):
        print(self.ktora)

prawa = Reka()
lewa = Reka()

lewa.start("LEWA")
prawa.start("PRAWA")

prawa.pokaz()
lewa.pokaz()
```

Jak utworzyć klasę w pythonie?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Wyjście z tego programu (wyjście zawsze będzie na oddzielnym slajdzie, żeby dać możliwość samodzielnej analizy kodu):

```
PRAWA
LEWA
```

Jak utworzyć klasę w pythonie?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Najprostsza klasa w pythonie:

```
class Nic:
    pass
```

Składa się z nagłówka: (class Nic:) i ciała (pass). Z reguły ciało jest trochę bardziej rozbudowane, nagłówek też bywa wyposażony w dodatkowe informacje(o tym później).

Jak utworzyć klasę w pythonie?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Czyli nagłówek klasy składa się (na razie) z trzech rzeczy:

- słowa kluczowego **class**,
- nazwy klasy,
- znaku dwukropka,

Nagłówek klasy Ręka:

```
class Reka:
```

Parę słów o nazwie klasy

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Nazwa klasy musi być zgodna ze standardem nadawania nazw w Pythonie
  - musi zaczynać się od litery lub podkreślenia i może składać się tylko z litery, podkreślenia i cyfry
- Ponadto, przewodnik stylu Pythona (do znalezienia w internecie "PEP 8") rekomenduje nazywanie klas wg notacji CamelCase
  - początek z wielkiej litery; wszystkie następne słowa powinny również rozpocząć się od wielkiej litery
- Osobiście zalecam, aby nazwa klasy składała się z 1, 2 najwyżej 3 słów.

Lepsze i gorsze nazwy klas

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

dobre nazwy:

```
class Bulgulator:
class CalendarManager:
class DatabaseConnector:
```

nieprawidłowe nazwy:

```
class BU!!!!:
class 1111BU:
```

złe nazwy

```
class ZmyslneUrzadzenieDoRadzeniaSobieWRozSyt:
class bulwaziemniaczana:
class NieWiem:
class Nic:
class N:
```

Pytanie - czemu to złe nazwy

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

nieprawidłowe nazwy:

```
class BU!!!!:
class 1111BU:
```

złe nazwy

```
class ZmyslneUrzadzenieDoRadzeniaSobieWRozSyt:
class bulwaziemniaczana:
class NieWiem:
class Nic:
class N:
```

Konwencja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Nazewnictwo nazw zmiennych jest swego rodzaju konwencją. Dzięki temu np. niektóre narzędzia mogą użyć nazwy klasy do wykonania weryfikacji poprawności kodu albo jego jakości. Niektóre mechanizmy języka działają również dzięki konwencjom.
Używanie niektórych metod, budowanie aplikacji wg pewnych wytycznych nazywamy konwencją.

Czemu używać konwencji?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- W pewnym sensie nasze klasy można rozpatrywać jako rozszerzenia Pythona.
- Chcemy, aby nasze klasy były podobne do wbudowanych klas Pythona, dzięki czemu różnice między językiem, biblioteką standardową i aplikacją są zminimalizowane.
- W językach gdzie istnieją pewne konwencje, warto ich używać.
  - Inni programiści są w stanie nie tylko łatwiej zrozumieć nasz kod, ale również łatwiej go poprawić lub coś pomóc.

Tworzymy obiekty

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Obiekty tworzymy przez podanie nazwy klasy i nawiasów okrągłych,
- W dalszej części nauczymy się robić klasy, których obiektom można ustawiać wartości z zewnątrz już w momencie tworzenia.

Tworzymy obiekty

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Ten obiekt się utworzy, ale nie powiążemy go z żadną zmienną, nie będzie nazwany.

```
Reka()
```

Ten obiekt się utworzy, i będzie się można do niego odwołać przy pomocy zmiennej prawa.

```
prawa = Reka()
```

To raczej rzadko będzie naszą intencją, dajemy nową nazwę.

```
Hand = Reka
```

I wtedy możemy tak napisać

```
left = Hand()
```

Tworzymy obiekty

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Tu utworzymy dwa obiekty nadając im od razu wartości początkowe (nasza klasa jeszcze tego nie umie).

```
prawa = Reka("PRAWA")
lewa = Reka("LEWA")
```

Możemy również tworzyć obiekty i dodawać je do listy:

```
rece=[Reka(), Reka()]
moje_rece = [Reka("PRAWA"), Reka("LEWA")]
```

Jak utworzyć klasę w pythonie?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Teraz ciało klasy, razem z nagłówkiem:

```
class Reka:
    def start(self, ktora):
        self.ktora=ktora
    def pokaz(self):
        print(self.ktora)
```

Metody

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- metody definiujemy jak zwykłe funkcje
  - z wcięciem względem klasy
- pierwszym parametrem każdej metody jest self (za chwilę o nim),
- dalej normalna lista parametrów jak przy funkcjach,
  - jeżeli któryś parametr jest listą i ma mieć domyślną wartość pustą, to podajemy None,
- Przy wywołaniu nie podajemy parametru self.

Kilka słów o self

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- jest to parametr, który musi mieć każda metoda
  - i to jako pierwszy
- nazwa self jest tylko konwencją,
  - ale chyba najbardziej przestrzeganą
- Python wstawi za nas ten parametr przy wywołaniu każdej z metod
- jest to referencja do bieżącej instancji naszej klasy,
- czyli możemy się odwołać przy pomocy self.cos do atrybutu coś
- czyli możemy się odwołać przy pomocy self.metoda() do innej (lub tej samej w której jesteśmy) metody z tej klasy.

Przykłady: metody i self

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

```
class Reka:
    def start(self, ktora):
        self.ktora=ktora

    def pokaz(self):
        print(self.ktora)

    def pokaz_ozdobiona(self):
        print("--------")
        self.pokaz()
        print("--------")
```

Metody - wywołanie

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Mając zdefiniowaną prostą klasę, utworzone jej obiekty:

```
class Reka:
    def start(self, ktora):
        self.ktora=ktora
    def pokaz(self):
        print(self.ktora)

prawa = Reka()
lewa = Reka()
```

Możemy dla tych obiektów wywoływać metody:

```
lewa.start("LEWA")
prawa.start("PRAWA")

prawa.pokaz()
lewa.pokaz()
```

Jak utworzyć klasę w pythonie?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Zmodyfikujmy część ciała klasy (zamiast start):

```
class Reka:
    def __init__(self, ktora):
        self.ktora=ktora
```

Metoda __init__ jest wywoływana bezpośrednio po utworzeniu instancji klasy. Może kusić, aby nazwać ją konstruktorem klasy, co nie jest prawdą. Metoda __init__ jest podobna do konstruktora (często __init__ jest pierwszą metodą pisaną w klasie), działa podobnie (jest pierwszym kodem wykonywanym w nowo utworzonej instancji klasy), a nawet podobnie brzmi (słowo "init" sugeruje, że to konstruktor). Nie jest to prawda, ponieważ obiekt jest utworzony przed wywołaniem metody __init__, i dostajemy poprawną referencję do świeżo utworzonego obiektu. Jednak __init__ w Pythonie, jest tym co najbardziej przypomina konstruktor, a pełni prawie taką samą rolę.

Czy trzeba pisać init?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

- Implementacja metody __init__() nie jest wymagana.
- Jeśli tego nie zrobimy, nie zostaną utworzone żadne zmienne instancji, gdy obiekt jest tworzony.
- W niektórych przypadkach to domyślne zachowanie jest akceptowalne.

Z initem lepiej

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Teraz można już napisać (jak chcieliśmy od początku):

```
prawa=Reka("PRAWA")
lewa=Reka("LEWA")
```

Dobra zasada dla inita

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Dobrą zasadą przy implementacji klas jest to, że w momencie zakończenia inicjalizacji klasa nadaje się już do użytku.

- Czyli wszystkie atrybuty są ustawione.
- Da się użyć dowolnej metody.
- W przypadku prostych i średnio złożonych klas radzę przestrzegać tej zasady,
- W przypadku złożonych klas...
  - ...pomyśleć nad ich uproszczeniem (lub podziałem).

Kod wyjściowy

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

```
class Reka:
    def __init__(self, ktora):
        self.ktora=ktora
    def pokaz(self):
        print(self.ktora)

prawa = Reka("PRAWA")
lewa = Reka("LEWA")

prawa.pokaz()
lewa.pokaz()
```

Kod wyjściowy - co wypisze

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Oczywiście:

```
PRAWA
LEWA
```

Możliwość dowolnego rozszerzania klas

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Mimo dobrej zasady (że klasa po inicjalizacji nadaje się do użytku) Python pozwala (a robił to jako jeden z pierwszych) na dodawanie atrybutów do klas z zewnątrz tychże klas. Ma to oczywiście przydatne zastosowania. W związku z tym warto poznać ten mechanizm (szczególnie, że w podobnej formie spopularyzował się on w wielu popularnych obecnie językach).

Rozszerzanie - jakie to proste

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Mając naszą klasę Ręka rozszerzymy ją z zewnątrz o pole kolor.

```
prawa = Reka("PRAWA")
lewa = Reka("LEWA")
prawa.kolor="Zielona"
lewa.kolor="Fioletowa"
prawa.pokaz()
lewa.pokaz()
```

Rozszerzanie - ale uwaga!

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Jaki będzie efekt działania:

```
prawa.pokaz()
lewa.pokaz()
```

Rozszerzanie - ale uwaga!

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Jaki będzie efekt działania:

```
PRAWA
LEWA
```

A chcieliśmy

```
Zielona PRAWA
Fioletowa LEWA
```

Co musimy dopisać?

Rozszerzanie - ale uwaga!

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Niestety, nie wolno nam tak zrobić:

```
    def pokaz(self):
        print(self.kolor + ' ' + self.ktora)
```

Chociaż w tym przykładzie zadziała:

```
Zielona PRAWA
Fioletowa LEWA
```

Rozszerzanie - ale uwaga!

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Niestety, nie wolno nam tak zrobić:

```
    def pokaz(self):
        print(self.kolor + ' ' + self.ktora)
```

To w tym przykładzie nie bardzo:

```
        inna_reka=Reka("INNA")
        inna_reka.pokaz()

AttributeError: 'Reka' object has no attribute '
    kolor'
```

Rozszerzanie - ale uwaga!

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Niestety, nie wolno nam tak zrobić:

```
    def pokaz(self):
        print(self.kolor + ' ' + self.ktora)
```

A w tym jeszcze gorzej:

```
        inna_reka=Reka("INNA")
        inna_reka.kolory="Malachitowy"
        inna_reka.pokaz()

AttributeError: 'Reka' object has no attribute '
    kolor'
```

Ktoś się pomylił, a my (autorzy metody pokaz) cierpimy, mimo, że mieliśmy dobre serce (dopisaliśmy zewnętrzne pole do naszej metody). Więc tak nie róbmy!

Rozszerzanie - to jak?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Albo użytkownik naszej klasy sam się obsłuży:

```
print(prawa.kolor, end=' ')
prawa.pokaz()
print(lewa.kolor, end=' ')
lewa.pokaz()
```

```
Zielona PRAWA
Fioletowa LEWA
```

Rozszerzanie - to jak?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Albo my (jako autorzy klasy) przewidzimy rozwój wypadków i skorzystamy np. z: __dict__:

```
prawa = Reka("PRAWA")
prawa.kolor="Zielona"
inna_reka=Reka("INNA")
>>> prawa.__dict__ --> {'kolor': 'Zielona', 'ktora':
     'PRAWA'}
>>> inna_reka.__dict__ --> {'ktora': 'INNA'}
```

i wtedy pokaz wyglądało by np. tak (o ile taki jest efekt oczekiwany przy braku pola):

```
    def pokaz(self):
        if "kolor" in self.__dict__:
            print(self.kolor, end=' ')
        print(self.ktora)
```

Co nam to wyświetli?

Rozszerzanie - to jak?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Nowe pokaz:

```
    def pokaz(self):
        if "kolor" in self.__dict__:
            print(self.kolor, end=' ')
        print(self.ktora)
```

Wyświetli:

```
Zielona PRAWA
INNA
```

O co chodzi z tymi podkreśleniami?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Były już dwie takie podejrzane nazwy: __init__ i __dict__ a to jeszcze nie koniec.

O co chodzi z tymi podkreśleniami?

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

O dunder, czyli zgodnie z `http://sjp.pl/dunder` dawniej, żartobliwie: piorun; dziś żywe w potocznych związkach frazeologicznych "niech cię (mnie, go itp.) dunder świśnie" albo "bodaj cię (mnie, go itp.) dunder świsnął" - niech przepadnie, niech cię (mnie, go itp.) piorun strzeli. Dunder - Ale nas interesuje angielskie znaczenie Double UNDERscore. Czyli __

O co chodzi z tymi podkreśleniami?

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Symbol ten ma dwa znaczenia:

- magiczne metody, jak __init__, __dict__ i za chwilę __len__
  - wywoływane w odpowiednich momentach, albo przydatne do realizacji pewnych elementów programów,
  - podkreślenia są z przodu i z tyłu,
- zmienne „prywatne”
  - zmienna o nazwie __nazwa w klasie Reka jest niedostepna przez odwołanie prawa.__nazwa
  - ale cały czas można się do niej odwołać przez : prawa._Reka__nazwa
  - oznacza to, że programista klasy ostrzega nas: Uwaga, wysokie napięcie, lepiej nie dotykać tej zmiennej. Chyba, że na własne ryzyko.
  - podkreślenia tylko z przodu

Długość ręki

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Teraz na przykładzie naszej klasy sprawdzimy jak można wykorzystać jedną ze specjalnych metod , czyli wspomniane już __len__. Dopiszemy ją do klasy Reka.

```
        def __len__(self):
            if self.ktora=="PRAWA":
                return 3
            else:
                return 4
print(len(prawa))
print(len(lewa))
```

W wyniku otrzymamy oczywiście:

```
3
4
```

Obiekty możemy wysyłać do funkcji

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Rozważmy przykład sprzed kilku slajdów, który aż się prosi o wdrożenie DRY (Don't Repeat Yourself):

```
print(prawa.kolor, end=' ')
prawa.pokaz()
print(lewa.kolor, end=' ')
lewa.pokaz()
```

Linie 1 od 3 i 2 od 4 różnią się od siebie tylko ...?

Obiekty możemy wysyłać do funkcji

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

... obiektem, z którego wywoływane są metody lub wyciągane atrybuty. Więc robimy funkcje, która jako parametr będzie miała obiekt:

```
def pokaz_reke(reka):
    print(reka.kolor, end=' ')
    reka.pokaz()
```

I wywołujemy ją dla obu obiektów.

```
pokaz_reke(prawa)
pokaz_reke(lewa)
```

Linii kodu jest o 1 więcej, ale z każdą kolejną "Ręką" się to niweluje. Najważniejsza jest jednak w tym wypadku czytelność kodu i możliwość operowania na wyższym poziomie abstrakcji (mniej szczegółów do ogarnięcia).

Obiekty możemy też przechowywać w listach

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Jak wspomnieliśmy, w ten sposób, przechowujemy obiekty nienazwane, tworzone tylko na potrzeby wpisania na listę:

```
rece=[Reka(), Reka()]
moje_rece=[Reka("LEWA"), Reka("PRAWA")]
```

Natomiast w ten sposób obiekty nazwane:

```
x = Reka()
y = Reka()
rece=[x, y]
```

Możliwe, choć niekoniecznie sensowne jest oczywiście mieszanie obydwu sposobów:

```
x = Reka()
y = Reka()
rece=[x, Reka(), Reka(), y]
```

Dokumentacja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Klasy i metody można dokumentować tak jak zwykłe funkcje:

```
class Reka:
    """Klasa do opisu reki
    mozna okreslac kolor, rodzaj itp."""
    def __init__(self, ktora):
        self.ktora=ktora
    def __len__(self):
        'Metoda␣podaje␣dlugosc␣reki'
        if self.ktora=="PRAWA":
            return 3
        else:
            return 4
    def pokaz(self):
        if "kolor" in self.__dict__:
            print(self.kolor, end='␣')
        print(self.ktora)
```

Dokumentacja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Opisy można obejrzeć wpisując help(nazwa_klasy)

```
        >>> help(Reka)
Help on class Reka in module __main__:
class Reka(builtins.object)
 |   Klasa do opisu reki
 |   mozna okreslac kolor, rodzaj itp.
 |   Methods defined here:
 |   __init__(self, ktora)
 |       Initialize self.  See help(type(self)) for
     accurate signature.
 |   __len__(self)
 |       Metoda podaje dlugosc reki
 |   pokaz(self)
 |   ----------------------
 |   Data descriptors defined here:
 |   ...
```

Umieszczanie klas w modułach

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Klasy można umieszczać w modułach, ale można je również wyposażyć np. w kod testujący, który będzie uruchamiany tylko przy uruchamianiu tego konkretnego modułu

```
class Reka:
        ....


def main():
    prawa = Reka("PRAWA")
    lewa = Reka("LEWA")
    prawa.pokaz()
    lewa.pokaz()

if __name__ == "__main__":
    main()
```

Klasy w funkcjach

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

```
def check_password(password, user_input,
                   checker=None):
    class DefaultPasswordChecker:
        def check(self, password, user_input):
            return password == user_input
    if not checker:
        checker = DefaultPasswordChecker()
    return checker.check(password, user_input)

haslo="ABCD1234"
print("Haslo prawidlowe: ")
print("tak" if check_password("ABCD1234", haslo)
    else "nie")
haslo="ABCD12345"
print("tak" if check_password("ABCD1234", haslo)
    else "nie")
```

Klasy w funkcjach - wyjście

Programowanie obiektowe

Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

```
Haslo prawidlowe:
tak
nie
```

Dostęp do zmiennych

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Zgodnie z przyjętą konwencją, można również poprzedzić nazwę atrybutu lub metody pojedynczym znakiem podkreślenia. Programiści Pythona zinterpretują to jako „to zmienna wewnętrzna, pomyśl trzy razy przed odwołaniem się do niej bezpośrednio”. Ale nie ma żadnego mechanizmu w Pythonie żeby powstrzymać ich przed dostępem, jeśli uważają, że leży to w ich najlepszym interesie, aby jednak się do tej zmiennej odwołać bezpośrednio. Jeśli się zastanowić, dlaczego mamy ich powstrzymywać? Możemy nie mieć pojęcia, jak w przyszłości będzie wykorzystywana nasza klasa.
W świecie (Monty) Pythona traktujemy się poważnie.

Jeszcze jedna magiczna funkcja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Funkcja __repr__ wywoływana wtedy kiedy jest potrzebna reprezentacja obiektu, dobrym zwyczajem jest jej implementacja w każdej klasie.
Wywołanie odbywa się bez użycia __repr__

```
print(prawa)
print(lewa)
print(inna)

<__main__.Reka object at 0x02DEDD30>
<__main__.Reka object at 0x02DEDD50>
<__main__.Reka object at 0x027D8590>
```

Jeszcze jedna magiczna funkcja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Implementacja __repr__

```
    def __repr__(self):
        napis=""
        if "kolor" in self.__dict__:
            napis = napis + self.kolor + ' '
        napis = napis + self.ktora
        return napis
```

Wywołanie z __repr__

```
print(prawa)
print(lewa)
print(inna)

PRAWA
LEWA
Zielona INNA
```

Jeszcze jedna magiczna funkcja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Podobna do __repr__ funkcja __str__, stosowana rzadziej, kiedy chcemy uzyskać wersję bardziej dla ludzi. Jeśli nie ma __str__, to używane jest __repr__.

```
class Demo:
    def __repr__(self):
        return 'demo'

>>> print(str(Demo()))
demo
>>> print(repr(Demo()))
demo
>>> print(Demo())
demo
```

Jeszcze jedna magiczna funkcja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Podobna do __repr__ funkcja __str__, stosowana rzadziej, kiedy chcemy uzyskać wersję bardziej dla ludzi. Jeśli nie ma __str__, to używane jest __repr__.

```
class Demo:
    def __str__(self):
        return 'demo'

>>> print(str(Demo()))
demo
>>> print(repr(Demo()))
<__main__.Demo object at 0x02869930>
>>> print(Demo())
demo
```

Jeszcze jedna magiczna funkcja

Programowanie obiektowe
Zagadnienia
Podstawowe
OOP
Obiekty
Klasy
tworzenie
Metody
self
rozszerzanie
Obiekty
dokumentacja
klasy w funkcjach
dostęp
repr

Podobna do __repr__ funkcja __str__, stosowana rzadziej, kiedy chcemy uzyskać wersję bardziej dla ludzi. Jeśli nie ma __str__, to używane jest __repr__.

```
class Demo:
    def __repr__(self):
        return 'demo_repr'
    def __str__(self):
        return 'demo_str'


>>> print(str(Demo()))
demo_str
>>> print(repr(Demo()))
demo_repr
>>> print(Demo())
demo_str
```