Akademia Pythona

VI Klasy i programowanie zorientowane obiektowo

KN Pythona - Kurs Pythona

KN Pythona wita na kursie Pythona.

Plan:

- Przeciążanie operatorów
- Projektowanie z użyciem klas

Przeciążanie operatorów

Przeciążanie operatorów:

- Przeciążanie operatorów pozwala klasom przechwytywać operacje Pythona.
- Klasy mogą przeciążać wszystkie operacje wyrażeń w Pythonie.
- Klasy mogą również przeciążać wbudowane operacje, jak wyświetlanie znaków, wywołania funkcji, dostęp do atrybutów itp.
- Przeciążanie operatorów pozwala klasom definiowanym przez użytkownika działać w sposób zbliżony do typów wbudowanych.
- Przeciążanie jest implementowane przez definiowanie metod klas o specjalnych nazwach.

Przeciążanie operatorów

number.py

```
class Number:
    def __init__(self, start):
        self.data = start

    def __sub__(self, other):
        return Number(self.data - other)
```

Przeciążanie operatorów

```
from number import Number

X = Number(5)
Y = X - 2
Y.data # 3
```

Przeciążanie operatorów

Metoda	Przeciąża	Wywoływana dla
init	Konstruktor	x = Klasa(args)
del	Destruktor	zwolnienie obiektu x
add	Operator +	x + y, x += y(bez iadd)
or	Bitowe OR	bit OR
repr, **str**	Wyświetlanie	str(x)

Przeciążanie operatorów

Metoda	Przeciąża	Wywoływana dla
call	Wywołania funkcji	x()
getattr	Kwalifikacja	x.undefined
setattr	Przypisanie atrybutu	x.b = y
delattr	Usuwanie atrybutu	del x.b

Przeciążanie operatorów

Metoda	Przeciąża	Wywoływana dla
getattribute	Przechwytywanie	x.b
getitem	Indeksowanie	x[a], x[i:j]
setitem	Przypisanie indeksu	x[a] = y
delitem	Usuwanie indeksu	del x[a]

Przeciążanie operatorów

Metoda	Przeciąża	Wywoływana dla
len	Długość	len(x), testy bool
bool	Testy logiczne	if x, bool(x)
lt, **gt**	Porównania	x $<$ y itd.
le, **ge**	Porównania	x $<=$ y itd.

Przeciążanie operatorów

Metoda	Przeciąża	Wywoływana dla
eq, **ne**	Porównania	x != y itd.
iter, **next**	Iteracja	for a in x
contains	Przynależność	a in S
enter, **exit**	Menadżer kontektu	with x as y:

Przeciążanie operatorów

```
class Indexer:
    def __getitem__(self, index):
        return index ** 2

X = Indexer()
X[2] # 4
X[3] # 9
```

Przeciążanie operatorów

```
L = [5, 6, 7, 8, 9]
L[2:4]
L[1:]
L[:-1]
L[::2]
```

Przeciążanie operatorów

```
L = [5, 6, 7, 8, 9]
L[slice(2, 4)]
L[slice(1, None)]
L[slice(None, 1)]
L[slice(None, None, 2)]
```

Przeciążanie operatorów

```
class Indexer:
    data = [5, 6, 7, 8, 9]
    def __getitem__(self, index):
        return self.data[index]
```

Iteracja po indeksie

```
class Stepper:
    def __getitem__(self, i):
        return self.data[i]
X = Stepper()
X.data = 'python'
for a in X:
    print(a) # p y t h o n
```

Przeciążanie operatorów

```
class Squares:
    def __init__(self, start, stop):
        self.value = start - 1
        self.stop = stop
    def __iter__(self):
        return self
    def __next__(self):
        if self.value == self.stop:
            raise StopIteration
        self.value += 1
        return self.value ** 2
```

Przeciążanie operatorów

```
x = Squares(1, 100)
for a in x:
    print(a)
```

Przeciążanie operatorów

```
def gssquares(start, stop):
    for i in range(start, stop + 1):
        yield i ** 2
```

Przeciążanie operatorów

```
class empty:
    def __getattr__(self, attrname):
        if attrname == 'age':
            return 40
        else:
            raise AttributeError(attrname)
```

Przeciążanie operatorów

```
class accesscontrol:
    def __setattr__(self, attr, value):
        if attr == 'age':
            self.__dict__[attr] = value
        else:
            raise AttributeError(attr)
```

Przeciążanie operatorów

```
class Calle:
    def __call__(self, *args, **kwargs):
        print('Wywołanie', args, kwargs)
c = Calle()
c('ab', data=12) # Wywołanie ('ab', ), {data: 12}
```

Projektowanie z użyciem klas

Kluczowe koncepcje:

- Dziedziczenie
- Polimorfizm
- Hermetyzacja

Projektowanie z użyciem klas

Sygnatury wywołań

```
class C:
    def meth(self, x):
        ...
    def meth(self, x, y, z):
        ...
```

Projektowanie z użyciem klas

```
class C:
    def meth(self, *args):
        if len(args) == 1:
            ...
        else:
            ...
```

Projektowanie z użyciem klas

Projetkowanie oparte na interfejsach

```
class C:
    def meth(self, x):
        x.operation()
```

Projektowanie z użyciem klas

employees.py

```
class Employee:
    def __init__(self, name, salary=0):
        self.name = name
        self.salary = salary
    def give_raise(self, percent):
        self.salary *= (1 + percent)
    def work(self):
        print(f'{self.name} pracuje.')
    def __repr__(self):
        return f'Pracownik: {self.name},
                wynagrodzenie: {self.salary}'
```

Projektowanie z użyciem klas

```
class Chef(Employee):
    def __init__(self, name):
        Employee.__init__(self, name, 50000)
    def work(self):
        print(f'{self.name} przygotowuje jedzenie.')
```

Projektowanie z użyciem klas

```
class Server(Employee):
    def __init__(self, name):
        Employee.__init__(self, name, 50000)
    def work(self):
        print(f'{self.name} obsługuje klienta.')
```

Projektowanie z użyciem klas

```
class PizzaRobot(Chef):
    def __init__(self, name):
        Chef.__init__(self, name)
    def work(self):
        print(f'{self.name} przygotowuje pizzę')
```

Projektowanie z użyciem klas

```
if __name__ == '__main__':
    bob = PizzaRobot('Robert')
    print(bob)
    bob.give_raise(.10)
    print(bob)
```

Projektowanie z użyciem klas

pizzashop.py

```
from employees import PizzaRobot, Server

class Customer:
    def __init__(self, name):
        self.name = name
    def order(self, server):
        print(f'{self.name} zamawia od {server}')
    def pay(self, server):
        print(f'{self.name} płaci {server}')
```

Projektowanie z użyciem klas

```
class Oven:
    def bake(self):
        print('Piec piecze')
```

Projektowanie z użyciem klas

```
class PizzaShop:
    def __init__(self):
        self.server = Server('Ernest')
        self.chef = PizzaRobot('Robert')
        self.oven = Oven()
```

Projektowanie z użyciem klas

```
...
    def order(self, name):
        customer = Customer(name)
        customer.order(self.server)
        self.chef.work()
        self.oven.bake()
        customer.pay(self.server)
```

Projektowanie z użyciem klas

```
if __name__ == '__main__':
    scene = PizzaShop()
    scene.order('Amadeusz')
    print('#' * 64)
    scene.order('Aleksander')
```

Delegacja

```
class Wrapper:
    def __init__(self, object):
        self.wrapped = object
    def __getattr__(self, attrname):
        print(f'Śledzenie: {attrname}')
        return getattr(self.wrapped, attrname)
```

Projektowanie z użyciem klas

Pseudoprywatne metody

```
class Klasa:
    def __method(self):
        ...
# Przekształcone na:
# _Klasa__method
```

Projektowanie z użyciem klas

Fabryki

```
def factory(cls, *args):
    return cls(args)
```

Projektowanie z użyciem klas

Inne zaganienia:

- Klasy abstrakcyjne
- Dekoratory
- Klasy typów
- Metody statyczne i metody klas
- Atrybuty zarządzane
- Metaklasy