# Wyklad11

December 20, 2019

**https://github.com/abarasinski/github001**

## 1 Klasy - ciąg dalszy

[2]:
```
class WektorN:
    """N-wymiarowy wektor"""
    def __init__(self, *args):
        setattr(self, "xlen", len(args))
        for idx, item in enumerate(args):
            setattr(self, "x{}".format(idx), item)

    def __str__(self):  # "nieformalna" reprezentacja obiektu: zwraca string
        wynik="Wektor: "
        for i in range(self.xlen):
            wynik+="{}, ".format(eval("self.x{}".format(i)))
        return wynik

    def __repr__(self):  # "nieformalna" reprezentacja obiektu: zwraca string
        wynik="["
        for i in range(self.xlen-1):
            wynik+="{}, ".format(eval("self.x{}".format(i)))
        wynik+="{}]".format(eval("self.x{}".format(i+1)))
        return wynik
```

[3]:
```
x_WN=WektorN(3,1,2,5,5,6,7)
```

[4]:
```
x_WN.__dict__
```

[4]:
```
{'xlen': 7, 'x0': 3, 'x1': 1, 'x2': 2, 'x3': 5, 'x4': 5, 'x5': 6, 'x6': 7}
```

[5]:
```
list(x_WN.__dict__.keys())
```

[5]:
```
['xlen', 'x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6']
```

[6]:
```
list(x_WN.__dict__.values())
```

[6]:
```
[7, 3, 1, 2, 5, 5, 6, 7]
```

[7]:
```
len(x_WN.__dict__)
```

[7]:
```
8
```

[8]:
```
class WektorN(object):
    """N-wymiarowy wektor"""

    def __new__(cls, *args):  #zawsze zaczynamu od __new__ jesli tylko jest zdefiniowane
        if str in list(map(type,args)):
            return print('Bledne dane')
        else:
            return object.__new__(cls)

    def __init__(self, *args):
        setattr(self, "xlen", len(args))   #potrzebne tylko ze wzgledow dydaktycznych
        for idx, item in enumerate(args):
            setattr(self, "x{}".format(idx), item)

    def __repr__(self):
        wynik="["
        for i in range(len(self.__dict__)-2):
            wynik+="{}, ".format(eval("self.x{}".format(i)))
        wynik+="{}]".format(eval("self.x{}".format(i+1)))
        return wynik

    def __getattr__(self, item):
            return None

    def __getattribute__(self, item): #najpierw __getattribute__ pozniej __getattr__
        if item=="xlen":
            print("Brak dostepu")
            raise AttributeError
        return object.__getattribute__(self,item)

    def __add__(self, w):
        temp=[]
        for i in range(len(self.__dict__)-1):
            temp.append(eval("self.x{}+w.x{}".format(i,i)))
        return WektorN(*temp)
```

[9]:
```
x_vec1=WektorN(3,1,2,5,5,6,7)
x_vec1
```

[9]: [3, 1, 2, 5, 5, 6, 7]

[10]:
```
x_vec2=WektorN(3,1,2,5,5,6,'7') #brak wyniku ze wzgledu na __new__
```

Bledne dane

[11]:
```
x_vec2
```

[12]:
```
type(x_vec2)
```

[12]: NoneType

[13]:
```
x_vec1.__dict__
```

[13]: {'xlen': 7, 'x0': 3, 'x1': 1, 'x2': 2, 'x3': 5, 'x4': 5, 'x5': 6, 'x6': 7}

[14]:
```
x_vec1.x0 #uruchamiamy __getattribute__
```

[14]: 3

[15]:
```
x_vec1.xlen #brak wyniku ze wzgledu na __getattribute__
```

Brak dostepu

[16]:
```
x_vec1.__dict__['xlen'] #ale tak dziala
```

[16]: 7

[17]:
```
x_vec1.x8 #brak wyniku ze wzgledu na __getattr__
```

Jakie to moze miec zastosowanie?

[18]:
```
x_vec1 + x_vec1
```

[18]: [6, 2, 4, 10, 10, 12, 14]

[19]:
```
x_vec3=WektorN(3,1,2,5,5,6)
x_vec3
```

[19]: [3, 1, 2, 5, 5, 6]

[20]:
```
x_vec1 + x_vec3
```

```
␣
↪---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call last)

<ipython-input-20-670e85f90b01> in <module>
----> 1 x_vec1 + x_vec3


<ipython-input-8-a762b5c56c51> in __add__(self, w)
     32         temp=[]
     33         for i in range(len(self.__dict__)-1):
---> 34             temp.append(eval("self.x{}+w.x{}".format(i,i)))
     35         return WektorN(*temp)
     36


<string> in <module>


TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'
```

[21]:
```
class WektorN(object):
    """N-wymiarowy wektor"""

    def __new__(cls, *args):
        if str in list(map(type,args)):
            return print('Bledne dane')
        else:
            return object.__new__(cls)

    def __init__(self, *args):
        setattr(self, "xlen", len(args))
        for idx, item in enumerate(args):
            setattr(self, "x{}".format(idx), item)

    def __repr__(self):
        wynik="["
        for i in range(len(self.__dict__)-2):
            wynik+="{}, ".format(eval("self.x{}".format(i)))
        wynik+="{}]".format(eval("self.x{}".format(i+1)))
        return wynik

    def __getattr__(self, item):
            return 0 #zamiast None

    def __getattribute__(self, item): #najpierw __getattribute__ pozniej␣
↪__getattr__
```

```
        if item=="xlen":
            print("Brak dostepu")
            raise AttributeError
        return object.__getattribute__(self,item)

    def __add__(self, w):
        temp=[]
        for i in range(len(self.__dict__)-1):
            temp.append(eval("self.x{}+w.x{}".format(i,i)))
        return WektorN(*temp)
```

[22]:
```
x_vec1=WektorN(3,1,2,5,5,6,7)
x_vec1.__dict__
```

[22]:
```
{'xlen': 7, 'x0': 3, 'x1': 1, 'x2': 2, 'x3': 5, 'x4': 5, 'x5': 6, 'x6': 7}
```

[23]:
```
x_vec3=WektorN(3,1,2,5,5,6)
x_vec3.__dict__
```

[23]:
```
{'xlen': 6, 'x0': 3, 'x1': 1, 'x2': 2, 'x3': 5, 'x4': 5, 'x5': 6}
```

[24]:
```
x_vec1 + x_vec3 #gdy nie mial x_vec3.x6 to wstawil sobie 0
```

[24]:
```
[6, 2, 4, 10, 10, 12, 7]
```

[25]:
```
x_vec1.__add__(x_vec3)
```

[25]:
```
[6, 2, 4, 10, 10, 12, 7]
```

[26]:
```
x_vec3.x9
```

[26]:
```
0
```

[27]:
```
class WektorN(object):
    """N-wymiarowy wektor"""

    def __new__(cls, *args):
        if str in list(map(type,args)):
            return print('Bledne dane')
        else:
            return object.__new__(cls)

    def __init__(self, *args):
        setattr(self, "xlen", len(args))
        for idx, item in enumerate(args):
            setattr(self, "x{}".format(idx), item)
```

```
        setattr(self, "norm", self.norma())              #tu mamy nowy atrybut !!
    ␣
↪#--------------------------------------------------------------------------------␣
↪
    def norma(self):
        temp=[]
        for i in range(self.__dict__['xlen']):
            temp.append(eval("self.x{}**2".format(i)))
        return pow(sum(temp),0.5)
    ␣
↪#--------------------------------------------------------------------------------␣
↪
    def __repr__(self):
        wynik="["
        for i in range(len(self.__dict__)-2):
            wynik+="{}, ".format(eval("self.x{}".format(i)))
        wynik+="{}]".format(eval("self.x{}".format(i+1)))
        return wynik

    def __getattr__(self, item):
            return 0

    def __getattribute__(self, item):
        if item=="xlen":
            print("Brak dostepu")
            raise AttributeError
        return object.__getattribute__(self,item)

    def __add__(self, w):
        temp=[]
        for i in range(len(self.__dict__)-2):
            temp.append(eval("self.x{}+w.x{}".format(i,i)))
        return WektorN(*temp)
```

[28]:
```
x_vec1=WektorN(3,1,2,5,5,6,7)
x_vec1.__dict__
```

[28]:
```
{'xlen': 7,
 'x0': 3,
 'x1': 1,
 'x2': 2,
 'x3': 5,
 'x4': 5,
 'x5': 6,
 'x6': 7,
 'norm': 12.206555615733702}
```

[29]:
```
pow(3*3+1*1+2*2+5*5+5*5+6*6+7*7,0.5)
```

[29]:
```
12.206555615733702
```

[30]:
```
x_vec1.x0=1
x_vec1.__dict__ #zmienil sie 'x0' ale 'norm' pozostala taka sama
```

[30]:
```
{'xlen': 7,
 'x0': 1,
 'x1': 1,
 'x2': 2,
 'x3': 5,
 'x4': 5,
 'x5': 6,
 'x6': 7,
 'norm': 12.206555615733702}
```

[31]:
```
class WektorN(object):
    """N-wymiarowy wektor"""

    def __new__(cls, *args):
        if str in list(map(type,args)):
            return print('Bledne dane')
        else:
            return object.__new__(cls)

    def __init__(self, *args):
        setattr(self, "xlen", len(args))
        for idx, item in enumerate(args):
            setattr(self, "x{}".format(idx), item)
        setattr(self, "norm", self.norma())                    #tu mamy nowy␣
↪atrybut !!
  ␣
↪#-------------------------------------------------------------------------------␣
↪
    def norma(self):
        temp=[]
        for i in range(self.__dict__['xlen']):
            temp.append(eval("self.x{}**2".format(i)))
        return pow(sum(temp),0.5)

    def __setattr__(self, name, value):                #a tak redefiniujemy␣
↪operacje self.name=value
        if name=='xlen':
            super().__setattr__(name, value)
        elif len(self.__dict__)==self.__dict__['xlen']+2:
            super().__setattr__(name, value+5)
```

```
            super().__setattr__("norm", self.norma())
            #self.norma()
        else:
            super().__setattr__(name, value)


␣
↪#--------------------------------------------------------------------------------␣
↪
    def __repr__(self):
        wynik="["
        for i in range(len(self.__dict__)-2):
            wynik+="{}, ".format(eval("self.x{}".format(i)))
        wynik+="{}]".format(eval("self.x{}".format(i+1)))
        return wynik

    def __getattr__(self, item):
            return 0

    def __getattribute__(self, item):
        if item=="xlen":
            print("Brak dostepu")
            raise AttributeError
        return object.__getattribute__(self,item)

    def __add__(self, w):
        temp=[]
        for i in range(len(self.__dict__)-1):
            temp.append(eval("self.x{}+w.x{}".format(i,i)))
        return WektorN(*temp)
```

[32]:
```
x_vec1=WektorN(3,1,2,5,5,6,7)
x_vec1.__dict__
```

[32]:
```
{'xlen': 7,
 'x0': 3,
 'x1': 1,
 'x2': 2,
 'x3': 5,
 'x4': 5,
 'x5': 6,
 'x6': 7,
 'norm': 12.206555615733702}
```

[33]:
```
x_vec1.x0=1
x_vec1.__dict__ #teraz przy zmienie 'x0' mamy zmiane 'norm'
```

[33]:
```
{'xlen': 7,
 'x0': 6,
 'x1': 1,
 'x2': 2,
 'x3': 5,
 'x4': 5,
 'x5': 6,
 'x6': 7,
 'norm': 13.2664991614216}
```

[34]:
```
pow(1*1+1*1+2*2+5*5+5*5+6*6+7*7,0.5)
```

[34]:
```
11.874342087037917
```

[ ]:
```

```