BTU ბიზნესისა და ტექნოლოგიების უნივერსიტეტი

python

პროგრამირება

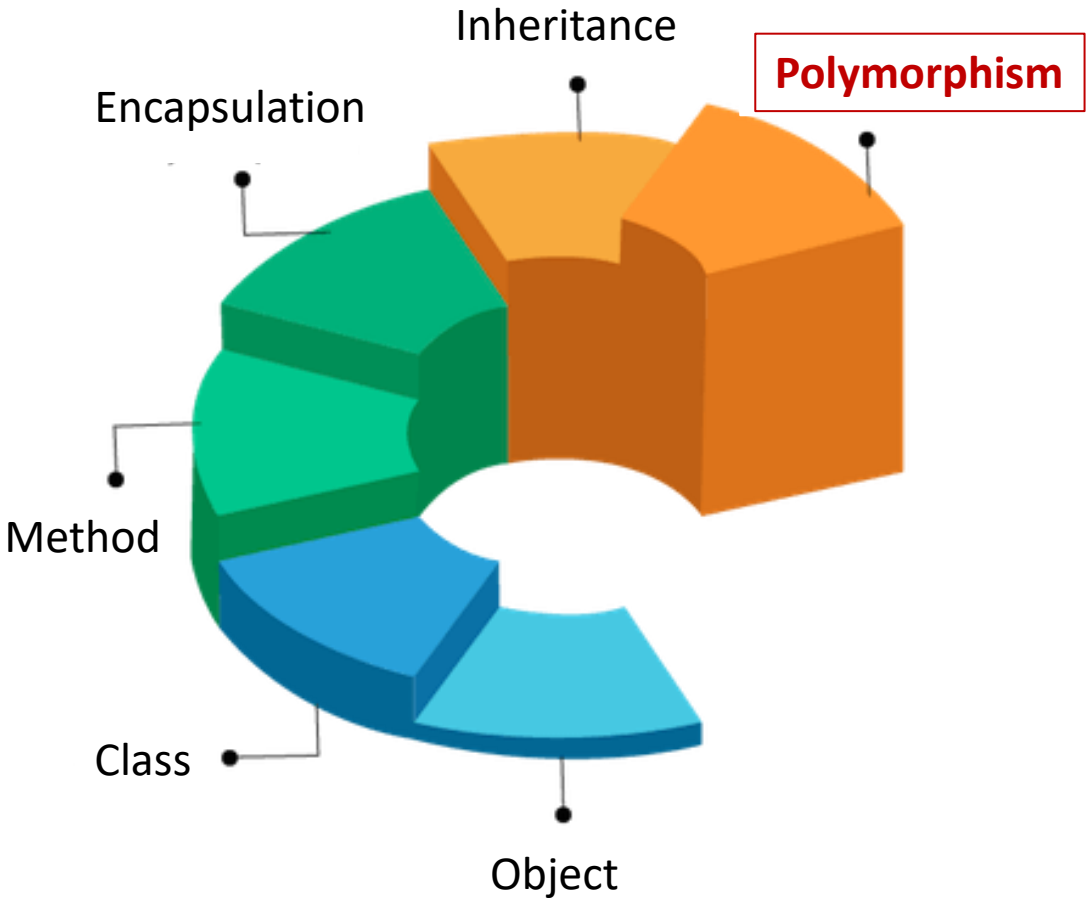
Python

ლექცია 3: პოლიმორფიზმი, მეთოდების გადატვირთვა, არითმეტიკული და შედარების ოპერატორების გადატვირთვა

ლიკა სვანაძე

lika.svanadze@btu.edu.ge

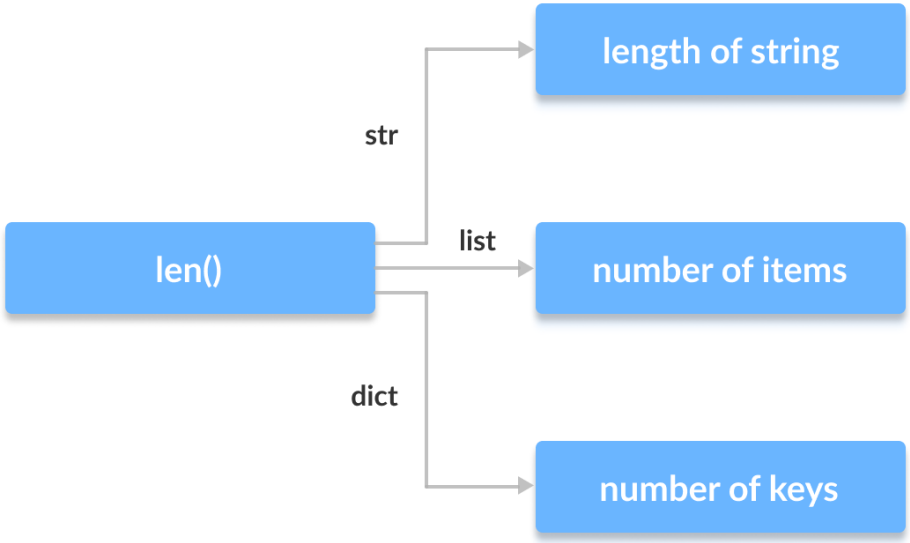
Python OOP system - პოლიმორფიზმი



დეფინიცია

* polymorphism ნიშნავს სხვადასხვა ფორმის პასუხს ერთიდაიგივე ფუნქციაზე (მეთოდზე). პოლიმორფიზმის გამოყენებით ერთი და იგივე ფუნქცია შესაძლებელია განსაზღვრული იყოს სხვადასხვა ტიპის მონაცემებისთვის (მაგ. int, float, str, list, სხვ.).
* ტიპზე დამოკიდებული მოქმედება (ქცევა) ცნობილია პოლიმორფიზმის სახელით.
* ობიექტზე ორიენტირებულ პროგრამირებაში პოლიმორფიზმი შეიძლება სხვადასხვა სახით იყოს წარმოდგენილი (იხილეთ მომდევნო სლაიდები).

პოლიმორფიზმი



პოლიმორფული კლასები

* კლასები პოლიმორფულია, როდესაც მათ აქვთ ერთი და იგივე სახელის მქონე ფუნქციები (მეთოდები), თუმცა შეიძლება ასრულებდნენ სხვადასხვა ოპერაციას. მეთოდის გამოძახება ხდება ერთნაირად, თუმცა სხვადასხვა კლასის ობიექტებისთვის.

მაგალითი

```
class Cat:
    def eat(self):
        print('Cat drinks milk')
    def walk(self):
        print('Cat walks 5km/h')

class Dog:
    def eat(self):
        print('Dog eats a bone')
    def walk(self):
        print('Dog walks 10km/h')


c1 = Cat()
d1 = Dog()
c1.eat()
d1.eat()
```

შედეგი:

```
Cat drinks milk
Dog eats a bone
```

პოლიმორფიზმი ფუნქციის გამოყენებით

შესაძლებელია შევქმნათ ფუნქცია კლასების გარეთ, რომელსაც პარამეტრად გადაეცემა ნებისმიერი ობიექტი და იძახებს ამ ობიექტისთვის კონკრეტულ მეთოდს (მაგ. eat მეთოდს). აღწერილი ფუნქციის გამოძახებისას პარამეტრად გადაეცემა ის ობიექტი, რომლისთვისაც გვინდა eat მეთოდის გამოძახება. მაგ. len ფუნქცია გამოიყენება როგორც სტრიქონისთვის, ასევე list, set ტიპის მონაცემებისთვის.

მაგალითი

```
class Cat:
    def eat(self):
        print('Cat drinks milk')
    def walk(self):
        print('Cat walks 5km/h')

class Dog:
    def eat(self):
        print('Dog eats a bone')
    def walk(self):
        print('Dog walks 10km/h')


def dinner(obj):
    obj.eat()

c1 = Cat()
d1 = Dog()
dinner(c1)
dinner(d1)
```

dinner ფუნქციის აღწერა, რომელსაც პარამეტრად გადაეცემა რაიმე ობიექტი, ფუნქციაში ხდება eat() მეთოდის შესრულება პარამეტრად გადაცემული ობიექტისთვის.

dinner(c1) - dinner ფუნქციის გამოძახება c1 ობიექტისთვის, რაც გულისხმობს რომ შესრულდება c1.eat() ოპერაცია, ხოლო dinner(d1) დროს შესრულდება d1.eat() ოპერაცია

შედეგი:

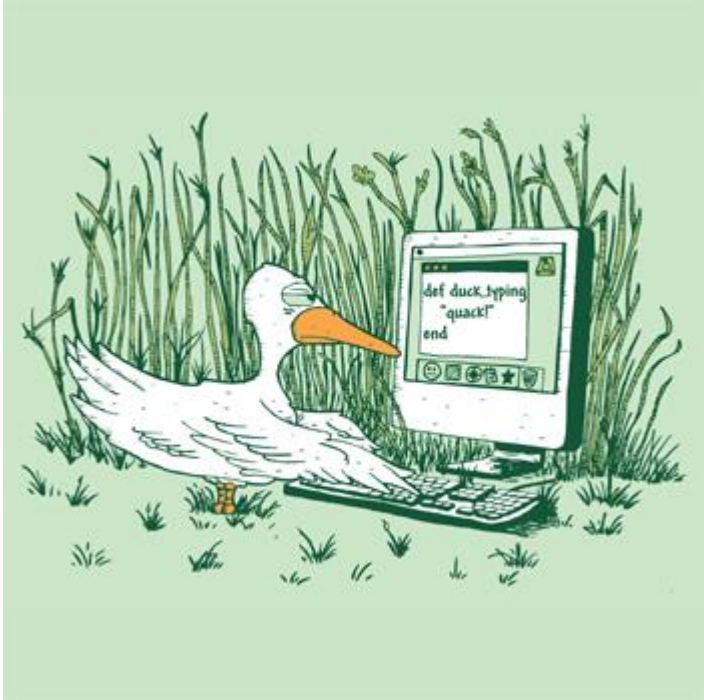
```
Cat drinks milk
Dog eats a bone
```

Duck Typing - Dynamic Typing

Be Pythonic

Not Pythonic

```
def dinner(obj):
    if isinstance(obj, Cat):
        obj.eat()
    else:
        print("This is not for Cat")
```



Duck Typing - Be Pythonic

```
def dinner(obj):
    obj.eat()
```

If it walks like a duck, swims like a duck and quacks like a duck, then it probably is a duck.

LBYL VS EAFP

- **LBYL** – "**L**ook **B**efore **Y**ou **L**eap" (მიმოიხედე სანამ გადახტები) – ***Not Pythonic***
- **EAFP** – "**E**asy to **A**sk **F**orgiveness than to get **P**ermission" - ***Pythonic***

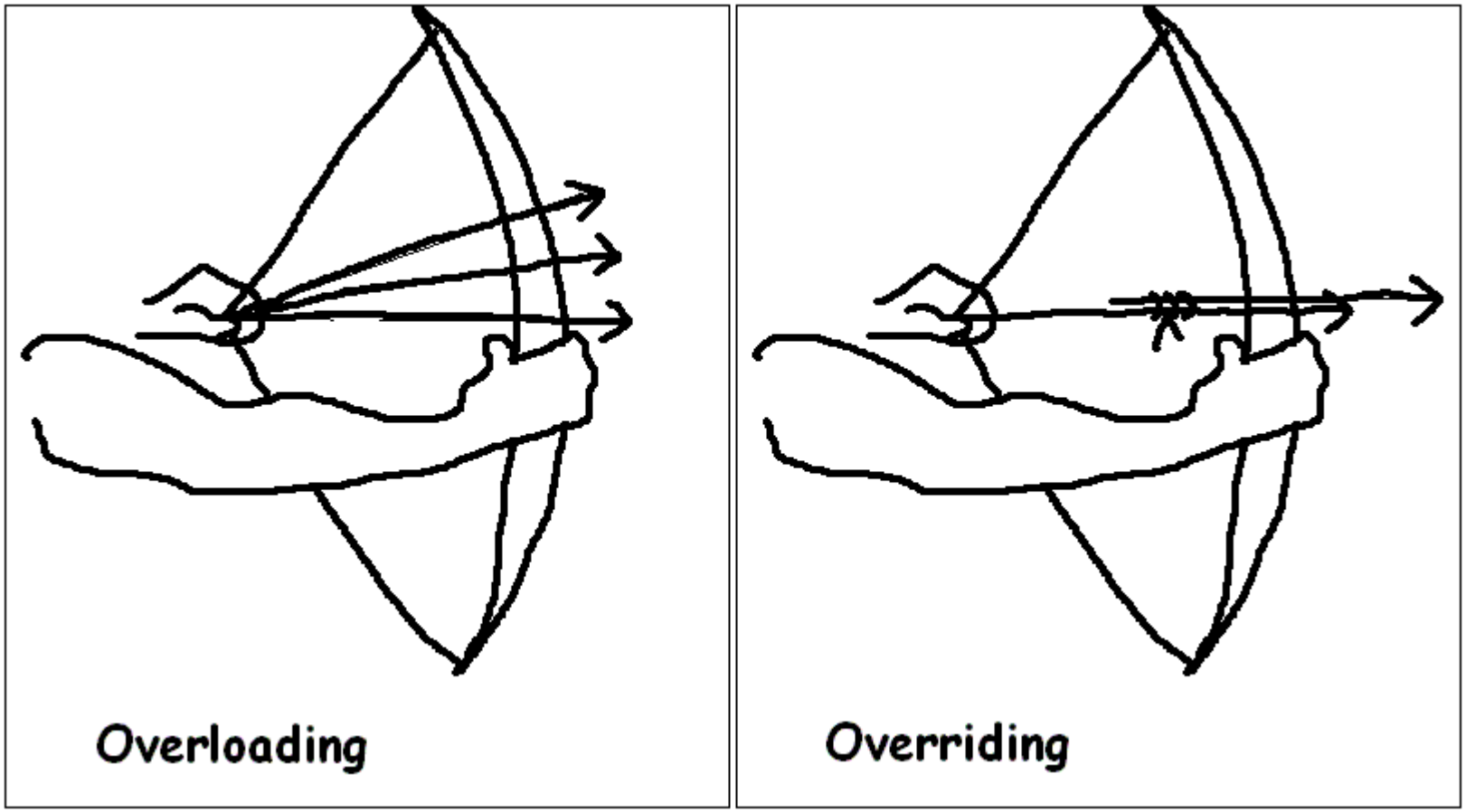
LBYL - Not Pythonic

```
def dinner(obj):
    if hasattr(obj, 'eat'):
        if callable(obj.eat):
            obj.eat()
```

EAFP - Be Pythonic

```
def dinner(obj):
    try:
        obj.eat()
    except AttributeError as msg:
        print(msg)
```

Method Overloading - მეთოდების გადატვირთვა



Method Overloading - მეთოდების გადატვირთვა

* მეთოდების გადატვირთვა ნიშნავს ერთი და იგივე სახელის მქონე მეთოდების გამოძახებას სხვადასხვა რაოდენობის ან სხვადასხვა ტიპის არგუმენტისთვის. სხვადასხვა პროგრამირების ენაში ეს საკითხი გადაწყვეტილია სხვადასხვანაირად. მაგ. C++-ში და სხვა ობიექტზე ორიენტირებულ ენებში შესაძლებელია განისაზღვროს ორი ან მეტი მეთოდი (ფუნქცია) ერთი და იგივე სახელი, მაგრამ სხვადასხვა რაოდენობის პარამეტრებით, ან სხვადასხვა ტიპის პარამეტრებით. გამოძახებისას კი გაეშვება ის მეთოდი, რომელშიც ემთხვევა პარამეტრების რაოდენობა და მათი ტიპები.

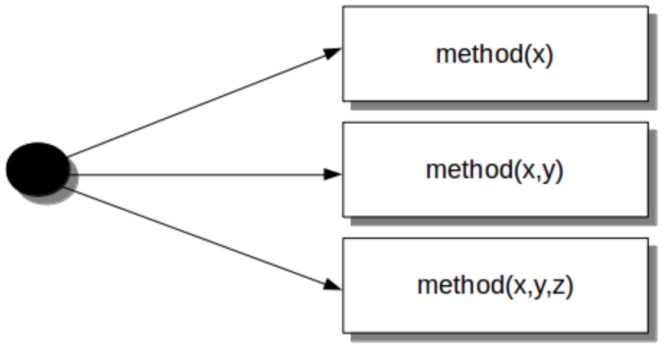
მეთოდების (ფუნქციების) გადატვირთვა C++-ში

Function Overloading Example

```
class Addition {
        public:
        void sum(int a, int b) {
                cout<<"a+b :"<<a+b;
        }
        void sum(int a, int b, int c) {
                cout<<"a+b+c :"<<a+b+c;
        }
};
int  main() {
        Addition obj;
        obj.sum(10, 20);
        cout<<endl;
        obj.sum(10, 20, 30);
}
```

ორივე ფუნქციას აქვს ერთი და იგივე სახელი და სხვადასხვა რაოდენობის პარამეტრი

სხვადასხვა ფუნქციის გამოძახება



Method Overloading - მეთოდების გადატვირთვა

* Python-ში **მეთოდების გადატვირთვა** შემდეგნაირად არის გადაწყვეტილი: შესაძლებელია მეთოდის შექმნა/აღწერა მხოლოდ ერთხელ ისე, რომ მისი გამოძახება მოხდეს სხვადასხვა გზით. მეთოდი შეგვიძლია გამოვიძახოთ სხვადასხვა რაოდენობა არგუმენტისთვის: არცერთი, ერთი, ორი ან მეტი პარამეტრით. იმისათვის რომ სხვადასხვა რაოდენობის პარამეტრებისთვის გამოვიძახოთ მეთოდი, საჭიროა პარამეტრებს მივუთითოთ None საწყისი (default) მნიშვნელობა. ხოლო, ფუნქციის შიგნით განხილული უნდა იყოს სხვადასხვა შესაძლო ვარიანტები.

მაგალითი

```
class Cat:
 def sayHi(self, name = None):
        if name is not None:
            print("Hi " + name)
        else:
            print("Hi X")

c1 = Cat()
c1.sayHi()
c1.sayHi(”Tom")
```

შედეგი:

```
Hi X
Hi Tom
```

ოპერატორების გადატვირთვა

* არითმეტიკული ოპერაციები შეიძლება გამოყენებული იქნას სხვადასხვა ტიპის მონაცემებისთვის (int, float, str, list, ა.შ). int, str, list და სხვა ტიპები წარმოადგენენ უკვე განსაზღვრულ კლასებს, რომლებსაც აქვთ არითმეტიკული ოპერაციების მხარდაჭერა. თითოეული კლასისთვის განსაზღვრულია მეთოდები, რომლის შესრულებაც ხდება კონკრეტული არითმეტიკული ოპერატორის გამოყენებისას.
* **ოპერატორების გადატვირთვა** გულისხმობს სასურველი მოქმედებების (მეთოდების) შესრულებას არითმეტიკული ოპერატორების გამოყენებისას.

```
# + და * ოპერატორი სხვადასხვა ტიპის მონაცემებთან
print(1 + 2)

# ორი სტრიქონის შეერთება + ოპერატორის გამოყენებით
print("BTU" + "University")

# ორი რიცხვის გამრავლება
print(3 * 4)

# სტრიქონის გამეორება * ოპერატორის გამოყენებით
print("BTU" * 4)

# ორი სიის გაერთიანება
print([1,4,3] + [2,3])
```

შედეგი:

```
3
BTUUniversity
12
BTUBTUBTUBTU
[1, 4, 3, 2, 3]
```

ოპერატორების გადატვირთვა

ახალი კლასის აღწერისას შესაძლოა გამოყენებული იყოს არითმეტიკულ ოპერაციებთან სამუშაო მეთოდები. მაგ. სხვადასხვა კლასისთვის + ოპერატორი შესაძლოა ასრულებდეს სხვადასხვა მოქმედებებს. Python-ში არსებობს ე.წ. magic (სპეციალური) ფუნქციები (მეთოდი), რომელთა შესრულება ხდება ავტომატურად, როგორც კი რომელიმე არითმეტიკული (ან შედარების) ოპერატორის გამოყენება მოხდება. მაგ. + ოპერატორის შესაბამისი magic მეთოდია __add__(), რომელშიც პროგრამისტს შეუძლია კონკრეტული მოქმედებების განსაზღვრა. __add__() მეთოდი ავტომატურად შესრულდება როცა + ოპერატორის გამოყენება მოხდება ამ კლასის ობიექტებზე. არსებობს სხვა წინასწარ განსაზღვრული magic ფუნქციები (მომდევნო სლაიდზე):

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "({},{})".format(self.x,self.y)

    def __add__(self, other):
        a = self.x + other.x
        b = self.y + other.y
        return Point(a,b)

p1 = Point(1,2)
p2 = Point(3,4)
print(p1)
print(p2)
p3 = p1 + p2
p3 = p1.__add__(p2) # ასრულებს იგივე მოქმედებას
p3 = Point.__add__(p1,p2) # ასრულებს იგივე მოქმედებას
print(p3)
```

შედეგი:

```
(1,2)
(3,4)
(4,6)
```

magic (Dunder) methods (1)

არითმეტიკული და შემოკვებული მინიჭების ოპერატორები			
ოპერატორი	**magic method**	**შესრულება**	**შესრულება**
+	__add__(self, other)	p1 + p2 ან p1 + 4	p1.__add__(p2)
-	__sub__(self, other)	p1 - p2	p1.__sub__(p2)
*	__mul__(self, other)	p1 * p2	p1.__mul__(p2)
**	__pow__(self, other)	p1 ** p2	p1.__pow__(p2)
/	__truediv__(self, other)	p1 / p2	p1.__truediv__(p2)
//	__floordiv__(self, other)	p1 // p2	p1.__floordiv__(p2)
%	__mod__(self, other)	p1 % p2	p1.__mod__(p2)
+=	__iadd__(self, other)	p1 += p2	p1.__iadd__(p2)
-=	__isub__(self, other)	p1 -= p2	p1.__isub__(p2)
*=	__imul__(self, other)	p1 *= p2	p1.__imul__(p2)
**=	__ipow__(self, other)	p1 **= p2	p1.__ipow__(p2)
/=	__itruediv__(self, other)	p1 /= p2	p1.__itruediv__(p2)
//=	__ifloordiv__(self, other)	p1 //= p2	p1.__ifloordiv__(p2)
%=	__imod__(self, other)	p1 %= p2	p1.__imod__(p2)

magic (Dunder) methods (2)

არითმეტიკული ოპერაციებისთვის ცალკე არის განხილული **right side** ოპერაციებთან სამუშაო ფუნქციები, როგორიცაა __radd__(), __rsub__(), __rmul__(), __rpow__(), ა.შ. მათი შესრულება ხდება მაშინ, როდესაც მეორე ოპერანდი არის ამ კლასის წარმომადგენელი, ხოლო პირველი ოპერანდი არ არის ამავე კლასის წარმომადგენელი.

```
p1 = Point(1,2)
p2 = Point(3,4)

p3 = p1 + p2  # შესრულდება __add__(instance, instance) მეთოდი: instance + instance
p3 = p3 + 3   # შესრულდება __add__(instance, noninstance) მეთოდი: instance + noninstance - იხილეთ მაგალითი საკლასო py ფაილში
p3 = 3 + p3   # შესრულდება __radd__(instance, noninstance) მეთოდი: noninstance + instance - იხილეთ მაგალითი საკლასო py ფაილში
```

magic (Dunder) methods (3)

Python-ში არსებობს სხვა ბევრი magic მეთოდი რომელიც გაიყენება კლასებთან სამუშაოდ. magic ფუნქციები იწყება და მთავრდება ორი ქვედა ტირით, მაგ. __init__(), __str__(), __add__(). Python-ში არსებული magic ფუნქციების ნახვა შესაძლებელია dir() ფუნქციის გამოყენებით. მაგ. dir(int) ან dir(Point)

ხშირად გამოყენებადი Dunder მეთოდები		
magic method	**აღწერა**	**გამოძახება**
__init__(self [,])	კონსტრუქტორი (ინიციალიზაცია - კლასის ატრიბუტების განსაზღვრა); მეთოდის შესრულება ხდება ავტომატურად კლასის ობიექტის შემოღების დროს.	p1 = Point(3,6)
__str__(self)	კლასის ობიექტის წარმოდგენა ტექსტურ ფორმატში; შესრულება ხდება ობიექტის გამოძახებისას (მაგ. ბეჭდვისას) ან str() ფუნქციის გამოყენებისას. მეთოდი ობიექტს წარმოადგენს ადვილად წასაკითხს (easy to read for users)	print(p1) str(p1)
__repr__(self)	კლასის ობიექტის წარმოდგენა მანქანურ ფორმატში; შესრულება ხდება ობიექტის გამოხებისას ტერმინალში ან repr() ფუნქციის გამოყენებისას. განკუთვნილია უმეტესად დეველოპერებისთვის	repr(p1)
__del__(self)	დესტრუქტორი (კლასის ობიექტის წაშლა); მეთოდის შესრულება ხდება კლასის გარეთ del ოპერაციის გამოყენებისას	del p1
__len__(self)	len() ფუნქცია გამოიყენება სხვადასხვა ტიპის მონაცემებთან, მაგ. სტრიქონებთან, სიებთან. ასევე შესაძლებელია გამოიყენოთ ნებისმიერი კლასის ობიექტებისთვის. ამისათვის განსაზღვრეთ შესაბამისი __len__() ფუნქცია კლასში თქვენთვის სასურველი ალგორითმით, რომელიც ავტომატურად შესრულდება ობიექტზე len ფუნქციის გამოყენებისას	len(p1)

magic (Dunder) methods (4)

შედარების ოპერატორების შესაბამისი Dunder მეთოდები			
ოპერატორი	**magic method**	შესრულება	შესრულება
<	__lt__()	p1 < p2	p1.__lt__(p2)
<=	__le__()	p1 <= p2	p1.__le__(p2)
>	__gt__()	p1 > p2	p1.__gt__(p2)
>=	__ge__()	p1 >= p2	p1.__ge__(p2)
==	__eq__()	p1 == p2	p1.__eq__(p2)
!=	__ne__()	p1 != p2	p1.__ne__(p2)

სავარჯიშო 1

- კლასის აღწერა: შექმენით კლასი Cat შემდეგი სამი მეთოდით: eat(), talk(), walk() რომლებიც ბეჭდავს შესაბამის ტექსტს ეკრანზე. მაგ. eat() მეთოდი ბეჭდავს **"Cat eats a milk”,** talk() მეთოდი ბეჭდავს **"Cat says miaww”,** walk() მეთოდი ბეჭდავს **"Cat can run 20km/h”.**

- კლასის აღწერა: დაამატეთ მეორე კლასი Dog, რომელსაც აქვს იგივე სახელწოდების მეთოდები eat(), talk(), walk(), თუმცა განსხვავებული მოქმედებებით. მაგ. eat() მეთოდი ბეჭდავს **" Dog eats a bone”,** talk() მეთოდი ბეჭდავს **"Dog says Aww”,** walk() მეთოდი ბეჭდავს **" Dog can run 40km/h”.**

- ობიექტის შექმნა: კლასის გარეთ შემოიტანეთ Cat() კლასის ობიექტი, ასევე შემოიტანეთ Dog() კლასის ობიექტი. გამოიძახეთ ორივე კლასის მეთოდები.

- გამოიძახეთ სამივე ფუნქცია ორივე ობიექტისთვის (უმჯობესია გამოიყენოთ for ციკლი რომელიც გაირბენს ორივე ობიექტისთვის შედგენილ tuple-ს).

სავარჯიშო 2

კლასის აღწერა: შექმენით კლასი Currency ატრიბუტებით value და unit. value მიუთითებს თანხის ოდენობას, ხოლო unit ვალუტას (მაგ. ”GEL”, “EUR”, “USD”). მოახდინეთ ატრიბუტების ინიციალიზაცია (კონსტრუქტორი). unit-ის მნიშვნელობა გაჩუმებით აიღეთ “GEL”. აღწერეთ კლასის შემდეგი მეთოდები:

* __str__() მეთოდი, რომელიც აბრუნებს ობიექტს შემდეგი ფორმატით: 2 ციფრი ათწილად ნაწილში და ვალუტის დასახელება. მაგ. 100.00 USD
* changeTo მეთოდი, რომელიც ითვლის თანხას სხვა ვალუტაში და აბრუნებს შედეგს. self ობიექტის გარდა, პარამეტრად გადაეცემა ვალუტის დასახელება, რომელშიც უნდა მოხდეს კონვერტაცია.
* განსაზღვრთ + ოპერატორი. თუ ოპერანდები არის ობიექტები, ითვლის მათ ჯამს. თუ შესაკრებების ვალუტა არის განსხვავებული, შედეგი უნდა იყოს ჩაწერილი პირველი შესაკრების ვალუტაში (მაგ 100 USD + 200 EUR, შედეგი უნდა იყოს USD-ში). განსაზღვრეთ შემთხვევა როდესაც მეორე შესაკრები შეიძლება იყოს მთელი ან ათწილადი რიცხვი (ვალუტის გარეშე), ამ შემთხევაში იგულისხმება რომ მეორე შესაკრები არის იმავე ვალუტაში რაც პირველი (მაგ. 100 USD + 300 = 400 USD). გაითვალისწინეთ შემთხვევა როდესაც მთელი ან ათწილად რიცხვი შეიძლება იყოს პირველი შესაკრები (მაგ. 300 + 100 USD = 400 USD).
* განსაზღვრეთ * ოპერატორი. გამრავლების ოპერაცია უნდა შესრულდეს როდესაც ერთი თანამამრავლი არის თანხა ხოლო მეორე სკალარული ტიპის (მთელი ან ათწილადი რიცხვი), სხვა შემთხევავაში უნდა გამოიტანოს TypeError შემდეგი ტექსტით: “**გამრავლების ოპერაცია უნდა შესრულდეს მხოლოდ მთელ ან ათწილად რიცხვზე”.** განსაზღვრეთ შემთხვევა, როდესაც თანამამრავლები არის შეცვლილი თანმიმდევრობით (მაგ. 3 * 100 EUR).
* განსაზღვრეთ > შედარების ოპერატორი.

შენიშვნა: სხვადასხვა ვალუტების კონვერტაციისთვის გჭირდებათ ცვლადი (dict ტიპის), რომელშიც შეინახავთ აღნიშნული ვალუტების კურსს ლართან მიმართებაში. მაგ. 1 USD = 2.7 GEL, 1 EUR = 3 GEL. ცვლადი უმჯობესია ჩაწეროთ კლასის სტატიკური ცვლადის სახით (__init__() მეთოდის ზემოთ).

პროგრამის გაშვების შედეგი იხილეთ შემდეგ სლაიდზე.

სავარჯიშო 2 (შედეგი)

```
>>> obj1 = Currency(400)
>>> print(obj1)
400.00 GEL
>>> changed_obj1 = obj1.changeTo("USD")
>>> print(changed_obj1)
148.15 USD
>>> obj2 = Currency(100, "USD")
>>> obj3 = Currency(200, "EUR")
>>> print(obj2 + obj3)
322.22 USD
>>> sum_obj = obj2 + 100
>>> print(sum_obj)
200.00 USD
>>> sum_obj = obj3 * 2
>>> print(sum_obj)
400.00 EUR
>>> sum_obj = 2 * obj3
>>> print(sum_obj)
400.00 EUR
```

სავარჯიშო 3

აღწერეთ ორი კლასი შემდეგი მონაცემების მიხედვით:

Person
ატრიბუტები: • name • deposit (default value = 1 000, მიუთითებს რამდენი აქვს ამჟამად თანხა დეპოზიტზე) • loan (default value=0, მიუთითებს რამდენი აქვს ამჟამად სესხი აღებული)

House
ატრიბუტები: • ID - ბინის საკადასტრო კოდი • price - ბინის ფასი • owner - სახლის მეპატრონე (Person ტიპის) • status - ახალი ბინის დამატებისას სტატუსი არის ყოველთვის 'გასაყიდი'

* გაითვალისწინეთ owner-ის მნიშვნელობა არის Person კლასის ობიექტი (instance).
* Person კლასში დაამატეთ __str__() მეთოდი, რომელიც დააბრუნებს პიროვნების სრულ ინფორმაციას.
* კლასების გარეთ შემოიტანეთ 2 ობიექტი Person კლასის (მაგ.ერთი მეპატრონე, მეორე მყიდველი). ასევე, შემოტანეთ კონკრეტული ბინა (House ტიპის, რომელსაც ამჟამინდელ მფლობელად (owner) გადასცემთ ერთ-ერთ შემოტანილ პირს).
* House კლასში დაამატეთ ბინის გაყიდვის ფუნქცია/მეთოდი, რომლის დროსაც პარამეტრად გადაეცემა მყიდველი. თუ მეტი პარამეტრი არ გადაეცემა, ამ დროს უნდა შესრულდეს ბინის გაყიდვის ოპერაცია (გამყიდველის deposit უნდა გაიზარდოს ბინის ღირებულებით, owner უნდა შეიცვალოს, status გახდეს "გაყიდული" და დაბეჭდოს შესაბამისი შეტყობინება). თუ ამ მეთოდის გამოძახებისას მყიდველის გარდა კიდევ ერთი პარამეტრი (რიცხვი) გადაეცა, მაშინ შესრულდეს ბინის სესხით გაყიდვის ოპერაცია, სადაც პარამეტრად გადაცემული რიცხვი მიუთითებს კლიენტის მიერ აღებული სესხის ოდენობას. ფუნქციამ კი უნდა შეასრულოს შემდეგი ოპერაციები: გამყიდველის deposit უნდა გაიზარდოს ბინის ღირებულებით, owner უნდა შეიცვალოს, status გახდეს "გაყიდულია სესხით", მყიდველის სესხის (loan) ატრიბუტის მნიშვნელობა უნდა გაიზარდოს შესაბამისად და დაბეჭდოს ბინის გაყიდვის შესაბამისი შეტყობინება.
* კლასის გარეთ მოახდინეთ აღწერილი ფუნქციის გამოძახება და დარწმუნდით მის სისწორეში.