

Óbudai Egyetem  
Kandó Kálmán Villamosmérnöki Kar  
Python  
OOP

Dr. Schuster György

2017. november 13.

# Alapfogalmak

Az OOP az előző módszertanokhoz képest a következő új tulajdonságokat vezette be:

# Alapfogalmak

Az OOP az előző módszertanokhoz képest a következő új tulajdonságokat vezette be:

- 1 egységbezárás (encapsulation),

# Alapfogalmak

Az OOP az előző módszertanokhoz képest a következő új tulajdonságokat vezette be:

- 1 **egységbezárás (encapsulation),**
- 2 **öröklődés (inheritance),**

# Alapfogalmak

Az OOP az előző módszertanokhoz képest a következő új tulajdonságokat vezette be:

- 1 **egységbezárás (encapsulation),**
- 2 **öröklődés (inheritance),**
- 3 **többalakúság (polimorphism).**

# Alapfogalmak

**Egységbezárás:** az adott *objektum* tartalmazza a szükséges adatokat és azokat a függvényeket, amelyek ezeket az adatokat kezelik.

# Alapfogalmak

**Egységbezárás:** az adott *objektum* tartalmazza a szükséges adatokat és azokat a függvényeket, amelyek ezeket az adatokat kezelik.

**Öröklődés:** az *osztályok tulajdonságokat* (adatokat) és *eljárásokat* (függvényeket) örökölhetnek egymástól.

# Alapfogalmak

**Egységbezárás:** az adott *objektum* tartalmazza a szükséges adatokat és azokat a függvényeket, amelyek ezeket az adatokat kezelik.

**Öröklődés:** az *osztályok tulajdonságokat* (adatokat) és *eljárásokat* (függvényeket) örökölhetnek egymástól.

**Többalakúság:** azonos néven többféle *eljárás* is létezhet.



# Alapfogalmak

**Egységbezárás:** az adott *objektum* tartalmazza a szükséges adatokat és azokat a függvényeket, amelyek ezeket az adatokat kezelik.

**Öröklődés:** az *osztályok tulajdonságokat* (függvényeket) örökölhetnek egymástól.

Vannak még fogalmak, *amiket nem tisztáztunk*, de fogjuk.

**Többalakúság:** azonos néven többféle *eljárás* is létezhet.

# Alapfogalmak

**Egységbezárás:** az adott *objektum* tartalmazza a szükséges adatokat és azokat a függvényeket, amelyek ezeket az adatokat kezelik.

**Öröklődés:** az *osztályok tulajdonságokat* (adatokat) és *eljárásokat* (függvényeket) örökölhetnek egymástól.

**Többalakúság:** azonos néven többféle *eljárás* is létezhet.

# Alapfogalmak

**Osztály (class):** a felhasználó által definiált prototípus, amely leírja azokat az *objektumokat*, amelyek ebbe a "csoportba" tartoznak.

# Alapfogalmak

**Osztály (class):** a felhasználó által definiált prototípus, amely leírja azokat az *objektumokat*, amelyek ebbe a "csoportba" tartoznak.

**Osztály változó (class variable):** olyan változó, amely egy példányban van az osztályban és az osztály *objektumai* elérhetik.

# Alapfogalmak

**Osztály (class):** a felhasználó által definiált prototípus, amely leírja azokat az *objektumokat*, amelyek ebbe a "csoportba" tartoznak.

**Osztály változó (class variable):** olyan változó, amely egy példányban van az osztályban és az osztály *objektumai* elérhetik.

**Objektum (object) (példány):** az osztály egy példánya "megvalósulása".

# Alapfogalmak

**Osztály (class):** a felhasználó által definiált prototípus, amely leírja azokat az *objektumokat*, amelyek ebbe a "csoportba" tartoznak.

**Osztály változó (class variable):** olyan változó, amely egy példányban van az osztályban és az osztály *objektumai* elérhetik.

**Objektum (object) (példány):** az osztály egy példánya "megvalósulása".

**Adat tag (data member):** az osztály, vagy az objektum saját változója.

# Alapfogalmak

**Osztály (class):** a felhasználó által definiált prototípus, amely leírja azokat az *objektumokat*, amelyek ebbe a "csoportba" tartoznak.

**Osztály változó (class variable):** olyan változó, amely egy példányban van az osztályban és az osztály *objektumai* elérhetik.

**Objektum (object) (példány):** az osztály egy példánya "megvalósulása".

**Adat tag (data member):** az osztály, vagy az objektum saját változója.

**Metódus (method):** az osztály, vagy az objektum saját függvénye.

# Python OOP, osztály, objektum

Egy példa kezdetnek:



# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

A futtató program.

```
#!/usr/bin/python3
class Lock:
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__ (self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

A leendő objektum belső változója (ez jelzi a névtérhez tartozást).

Paraméter(ek).



# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

A leendő objektum belső változója (ez jelzi a névtérhez tartozást).

Paraméter(ek).

A névtér az objektum.

Az objektum belső változója.

Az átadott paraméter.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

A leendő objektum belső változója (ez jelzi a névtérhez tartozást).

Paraméter(ek).

A névtér az objektum.

Az objektum belső változója.

Az átadott paraméter.

Egy metódus, a **self**-et át kell adni.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'

lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

A leendő objektum belső változója (ez jelzi a névtérhez tartozást).

Paraméter(ek).

A névtér az objektum.

Az objektum belső változója.

Az átadott paraméter.

Egy metódus, a **self**-et át kell adni.

További metódusok.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

A leendő objektum belső változója (ez jelzi a névtérhez tartozást).

Paraméter(ek).

A névtér az objektum.

Az objektum belső változója.

Az átadott paraméter.

Egy metódus, a **self**-et át kell adni.

További metódusok.

Az objektum(példány) létrehozása.

Az **\_\_init\_\_** függvény ekkor fut le.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

A leendő objektum belső változója (ez jelzi a névtérhez tartozást).

Paraméter(ek).

A névtér az objektum.

Az objektum belső változója.

Az átadott paraméter.

Egy metódus, a **self**-et át kell adni.

További metódusok.

Az objektum(példány) létrehozása.

Az **\_\_init\_\_** függvény ekkor fut le.

További, az objektumhoz tartozó függvények hívása.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

A futtató program.

Az osztály kulcsszava.

Az osztály neve.

Egy belső függvény lesz.

Egy speciális inicializáló függvény (*majdnem konstruktor*).

A leendő objektum belső változója (ez jelzi a névtérhez tartozást).

Paraméter(ek).

A névtér az objektum.

Az objektum belső változója.

Az átadott paraméter.

Egy metódus, a **self**-et át kell adni.

További metódusok.

Az objektum(példány) létrehozása.

Az **\_\_init\_\_** függvény ekkor fut le.

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python3
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

# Python OOP, osztály, objektum

Egy példa kezdetnek: egy lakat.

```
#!/usr/bin/python3
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
lck1 = Lock('closed')
lck1.status()
lck1.open()
lck1.status()
lck1.close()
lck1.status()
```

Képernyő

```
closed
open
closed
```



# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.

# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.  
Tehát minden objektumra az osztályban.

# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.  
Tehát minden objektumra az osztályban.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.material='bronze'
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.  
Tehát minden objektumra az osztályban.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'

lck1 = Lock('closed')
lck2 = Lock('closed')

print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.material='bronze'
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

Ez már ismert.

# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.  
Tehát minden objektumra az osztályban.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.material='bronze'
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

Ez már ismert.  
Az osztály változó.

Képernyő

# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.  
Tehát minden objektumra az osztályban.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.material='bronze'
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

Ez már ismert.

**Az osztály változó.**

**Az osztály változó nyomtatása.**

Képernyő

```
steel
steel
steel
```

# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.  
Tehát minden objektumra az osztályban.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.material='bronze'
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

Ez már ismert.

**Az osztály változó.**

**Az osztály változó nyomtatása.**

**Az osztály változó új értéke,  
és nyomtatása.**

Képernyő

```
steel
steel
steel
bronze
bronze
bronze
```

# Python OOP, osztály változó

Az osztály változó az egész osztályra vonatkozik.  
Tehát minden objektumra az osztályban.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.material='bronze'
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

Ez már ismert.

**Az osztály változó.**

**Az osztály változó nyomtatása.**

**Az osztály változó új értéke,  
és nyomtatása.**

Képernyő

```
steel
steel
steel
bronze
bronze
bronze
```



# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

A program (hiányos).

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st

    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

A program (hiányos).  
Eddig semmi új.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st

    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

A program (hiányos).  
Eddig semmi új.  
Ide jönnek az objektum függvényei.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        : 🐘
lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

Sajnos nem fértek ki,  
de az előző példákban láthatók.

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

A program (hiányos).

Eddig semmi új.

Ide jönnek az objektum függvényei.

A következő függvény osztályfüggvény lesz.

Az osztályfüggvény.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st

    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st

    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

A program (hiányos).

Eddig semmi új.

Ide jönnek az objektum függvényei.

A következő függvény osztályfüggvény lesz.

Az osztályfüggvény.

A `lck` paraméter az osztály "névtere". Ez lehet bármi, csak `self` és foglalt kulcsszó nem.

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st

    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

A program (hiányos).

Eddig semmi új.

Ide jönnek az objektum függvényei.

A következő függvény osztályfüggvény lesz.

Az osztályfüggvény.

A `lck` paraméter az osztály "névtére". Ez lehet bármi, csak `self` és foglalt kulcsszó nem.

Az osztály paraméter nyomtatása.

Képernyő

```
steel
steel
steel
```



# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :
lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

A program (hiányos).

Eddig semmi új.

Ide jönnek az objektum függvényei.

A következő függvény osztályfüggvény lesz.

Az osztályfüggvény.

A `lck` paraméter az osztály "névtere". Ez lehet bármi, csak `self` és foglalt kulcsszó nem.

Az osztály paraméter nyomtatása.

Az osztályfüggvény hívása.

Képernyő

```
steel
steel
steel
```

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self,st):
        self.st=st
    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :
lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

A program (hiányos).

Eddig semmi új.

Ide jönnek az objektum függvényei.

A következő függvény osztályfüggvény lesz.

Az osztályfüggvény.

A `lck` paraméter az osztály "névtere". Ez lehet bármi, csak `self` és foglalt kulcsszó nem.

Az osztály paraméter nyomtatása.

Az osztályfüggvény hívása.

Az osztály paraméter nyomtatása.

Képernyő

```
steel
steel
steel
bronze
bronze
bronze
```

# Python OOP, osztály függvény

Az osztályváltozóhoz hasonlóan lehet osztályfüggvényt is létrehozni.  
Ebből a függvényből szintén csak egy van.

```
#!/usr/bin/python3
class Lock:
    material='steel'
    def __init__(self, st):
        self.st=st

    @classmethod
    def materialchk(lck):
        if lck.material=='steel':
            lck.material='bronze'
        else:
            lck.material='steel'
        :

lck1 = Lock('closed')
lck2 = Lock('closed')
print(lck1.material)
print(lck2.material)
print(Lock.material)
Lock.materialchk()
print(lck1.material)
print(lck2.material)
print(Lock.material)
```

A program (hiányos).

Eddig semmi új.

Ide jönnek az objektum függvényei.

A következő függvény osztályfüggvény lesz.

Az osztályfüggvény.

A `lck` paraméter az osztály "névtere". Ez lehet bármi, csak `self` és foglalt kulcsszó nem.

Az osztály paraméter nyomtatása.

Az osztályfüggvény hívása.

Az osztály paraméter nyomtatása.

Képernyő

```
steel
steel
steel
bronze
bronze
bronze
```

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.  
A példa egy ablak.

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paraméterei és  
függvényei

POS

$\begin{pmatrix} x \\ y \end{pmatrix}$

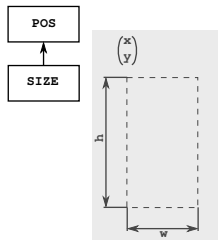
Az ablak bal felső sarkának koordinátái:  
**POS:  $x$ ,  $y$ .**

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paraméterei és  
függvényei



Az ablak bal felső sarkának koordinátái:

**POS:**  $x$ ,  $y$ .

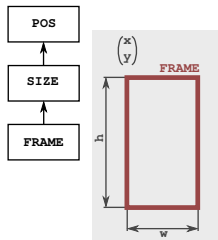
Az ablak geometriai méretei: **SIZE:**  $w$ ,  $h$ .

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paramétereit és  
függvényeit



Az ablak bal felső sarkának koordinátái:

**POS:**  $x$ ,  $y$ .

Az ablak geometriai méretei: **SIZE:**  $w$ ,  $h$ .

Az ablak kerete: **FRAME:** `fput()`.

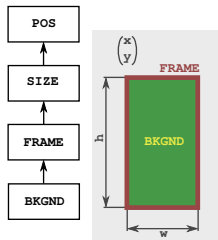


# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paramétereit és  
függvényeit



Az ablak bal felső sarkának koordinátái:

**POS:**  $x, y$ .

Az ablak geometriai méretei: **SIZE:**  $w, h$ .

Az ablak kerete: **FRAME:** `fput()`.

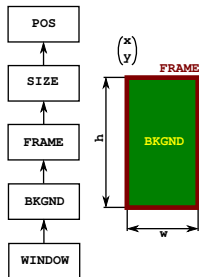
Az ablak háttere: **BKGND:** `bput()`.

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paramétereit és  
függvényeit



Az ablak bal felső sarkának koordinátái:

**POS:**  $x$ ,  $y$ .

Az ablak geometriai méretei: **SIZE:**  $w$ ,  $h$ .

Az ablak kerete: **FRAME:** `fput()`.

Az ablak háttere: **BKGND:** `bput()`.

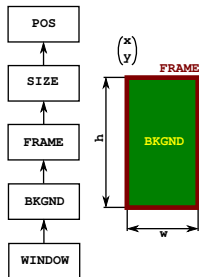
Az ablak: **WINDOW:** `wput()`.

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paramétereit és  
függvényeit



Az ablak bal felső sarkának koordinátái:

**POS:**  $x$ ,  $y$ .

Az ablak geometriai méretei: **SIZE:**  $w$ ,  $h$ .

Az ablak kerete: **FRAME:** `fput()`.

Az ablak háttere: **BKGND:** `bput()`.

Az ablak: **WINDOW**

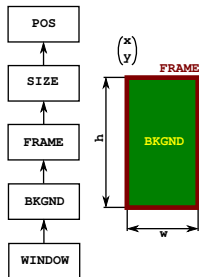
Ez egy mintapélda.  
Senki sem csinál így ablakot.

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paramétereit és  
függvényeit



Az ablak bal felső sarkának koordinátái:

**POS:**  $x$ ,  $y$ .

Az ablak geometriai méretei: **SIZE:**  $w$ ,  $h$ .

Az ablak kerete: **FRAME:** `fput()`.

Az ablak háttere: **BKGND:** `bput()`.

Az ablak: **WINDOW:** `wput()`.

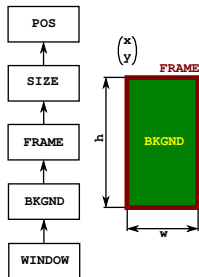
A grafikát csak "imitáljuk".  
Egyenlőre!

# Python OOP, öröklődés

A python képes az osztályok közötti öröklődésre.

A példa egy ablak.

Az ablak paramétereit és  
függvényeit



Az ablak bal felső sarkának koordinátái:

**POS:**  $x$ ,  $y$ .

Az ablak geometriai méretei: **SIZE:**  $w$ ,  $h$ .

Az ablak kerete: **FRAME:** `fput()`.

Az ablak háttere: **BKGND:** `bput()`.

Az ablak: **WINDOW:** `wput()`.

# Python OOP, öröklődés

Részletesen:

# Python OOP, öröklődés

Részletesen:



```
class POS:
    def __init__(s):
        s.x=0
        s.y=0
        :
```

A POS osztály definíciója.

# Python OOP, öröklődés

Részletesen:



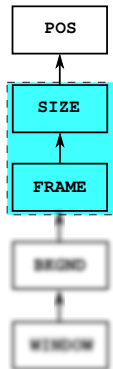
```
class SIZE(POS):  
    def __init__(s):  
        s.w=0  
        s.h=0
```

A **POS** osztály definíciója.  
A **SIZE** osztály  
definíciója, amely örököl  
a **POS** osztálytól.



# Python OOP, öröklődés

Részletesen:



```
class FRAME(SIZE):  
    def __init__(s):  
        pass  
    def fput(s):  
        print("f")  
        print(s.x, s.y, s.w, s.h)  
    :
```

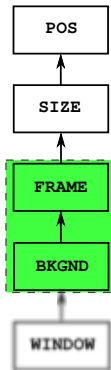
A **POS** osztály definíciója.

A **SIZE** osztály definíciója, amely örököl a **POS** osztálytól.

A **FRAME** osztály definíciója, amely örököl a **SIZE** osztálytól.

# Python OOP, öröklődés

Részletesen:



```
class BKGND(FRAME):  
    def __init__(s):  
        pass  
    def bput(s):  
        print("b")  
        print(s.x, s.y, s.w, s.h)  
    :
```

A **POS** osztály definíciója.

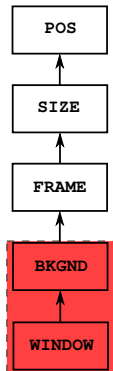
A **SIZE** osztály definíciója, amely örököl a **POS** osztálytól.

A **FRAME** osztály definíciója, amely örököl a **SIZE** osztálytól.

A **BKGND** osztály definíciója, amely örököl a **FRAME** osztálytól.

# Python OOP, öröklődés

Részletesen:



```
class WINDOW(BKGND):  
    def __init__(s, x, y, w, h):  
        s.x=x  
        s.y=y  
        s.w=w  
        s.h=h  
    def wput(s):  
        s.bput()  
        s.fput()
```

A **POS** osztály definíciója.

A **SIZE** osztály definíciója, amely örököl a **POS** osztálytól.

A **FRAME** osztály definíciója, amely örököl a **SIZE** osztálytól.

A **BKGND** osztály definíciója, amely örököl a **FRAME** osztálytól.

A **WINDOW** osztály definíciója, amely örököl a **BKGND** osztálytól és hívja a **BKGND** **bput** és a **FRAME** **fput** függvényeit.

# Python OOP, öröklődés

Részletesen:

Használjuk az így elkészült osztályt!

```
⋮  
w=WINDOW(10,20,30,40)  
w.wput()
```

Képernyő



# Python OOP, öröklődés

Részletesen:

Használjuk az így elkészült osztályt!

```
⋮  
w=WINDOW(10,20,30,40)  
w.wput()
```

Az előzmények. Az osztály  
definíciók természetesen  
megelőzik a használatot.

Képernyő

# Python OOP, öröklődés

Részletesen:

Használjuk az így elkészült osztályt!

```
⋮  
w=WINDOW(10,20,30,40)  
w.wput()
```

Az előzmények. Az osztály  
definíciók természetesen  
megelőzik a használatot.  
A példány létrehozása és  
paraméterezése.

Képernyő

# Python OOP, öröklődés

Részletesen:

Használjuk az így elkészült osztályt!

```
⋮  
w=WINDOW(10,20,30,40)  
w.wput()
```

Az előzmények. Az osztály  
definíciók természetesen  
megelőzik a használatot.

A példány létrehozása és  
paraméterezése.

A `wput` függvény hívása. Ez  
hívja a `bput` "b" és a `fput` "f"  
függvényeket.

## Képernyő

```
b  
10 20 30 40  
f  
10 20 30 40
```

# Python OOP, többszörös öröklődés

Lehet-e egy osztálynak több őse?



# Python OOP, többszörös öröklődés

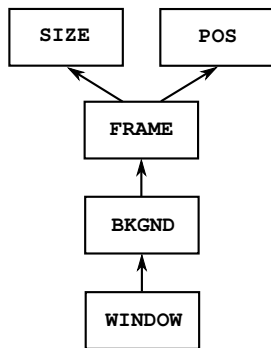
Lehet-e egy osztálynak több őse?

Igen lehet.

# Python OOP, többszörös öröklődés

Lehet-e egy osztálynak több őse?

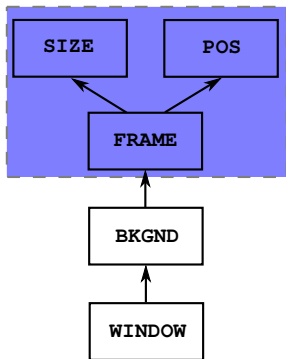
Igen lehet.



# Python OOP, többszörös öröklődés

Lehet-e egy osztálynak több őse?

Igen lehet.



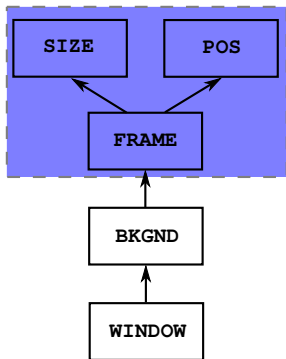
```
class POS:
    def __init__(s):
        s.x=0
        s.y=0
        :
```

A POS osztály  
definíciója.

# Python OOP, többszörös öröklődés

Lehet-e egy osztálynak több őse?

Igen lehet.



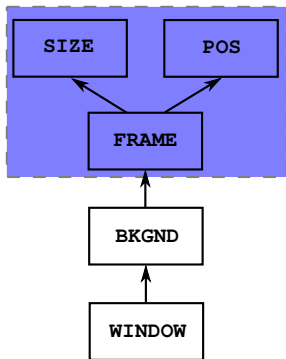
```
class SIZE:  
    def __init__(s):  
        s.w=0  
        s.h=0  
    :
```

A **POS** osztály  
definíciója.  
A **SIZE** osztály  
definíciója (nincs  
őse).

# Python OOP, többszörös öröklődés

Lehet-e egy osztálynak több őse?

Igen lehet.



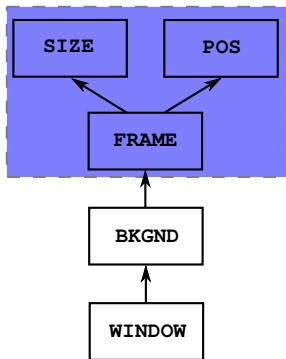
```
class FRAME(POS, SIZE):  
    def __init__(s):  
        pass  
    def fput(s):  
        print("f")  
        print(s.x, s.y, s.w, s.h)
```

A **POS** osztály definíciója.  
A **SIZE** osztály definíciója (nincs őse).  
A **FRAME** osztály definíciója. **Két őse van.**

# Python OOP, többszörös öröklődés

Lehet-e egy osztálynak több őse?

Igen lehet.



```
class FRAME(POS, SIZE):  
    def __init__(s):  
        pass  
    def fput(s):  
        print("f")  
        print(s.x, s.y, s.w, s.h)  
    :
```

A **POS** osztály definíciója.  
A **SIZE** osztály definíciója (nincs őse).  
A **FRAME** osztály definíciója. **Két őse van.**  
Minden más maradt a régiben.

# Python OOP, tartalmazás reláció

Ez a fogalom kicsit hasonlít az öröklődéshez, de működésében inkább az egységbezáráshoz köthető.

# Python OOP, tartalmazás reláció

Ez a fogalom kicsit hasonlít az öröklődéshez, de működésében inkább az egységbezáráshoz köthető.

Ez a reláció azt jelenti, hogy egy objektum több más objektumot tartalmaz (mint alkatrészeket).



# Python OOP, tartalmazás reláció

Ez a fogalom kicsit hasonlít az öröklődéshez, de működésében inkább az egységbezáráshoz köthető.

Ez a reláció azt jelenti, hogy egy objektum több más objektumot tartalmaz (mint alkatrészeket).

Kétfélét különböztetünk meg, ezek:

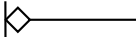
# Python OOP, tartalmazás reláció

Ez a fogalom kicsit hasonlít az öröklődéshez, de működésében inkább az egységbezáráshoz köthető.

Ez a reláció azt jelenti, hogy egy objektum több más objektumot tartalmaz (mint alkatrészeket).

Kétfélét különböztetünk meg, ezek:

- 1 gyenge tartalmazás, ahol az "alkatrész" nélkül is "működik" az objektum,

jele: 

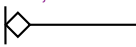
# Python OOP, tartalmazás reláció

Ez a fogalom kicsit hasonlít az öröklődéshez, de működésében inkább az egységbezáráshoz köthető.

Ez a reláció azt jelenti, hogy egy objektum több más objektumot tartalmaz (mint alkatrészeket).

Kétfélet különböztetünk meg, ezek:

- 1 gyenge tartalmazás, ahol az "alkatrész" nélkül is "működik" az objektum,

jеле: 

- 2 erős tartalmazás, ahol az objektum nem működik az "alkatrész" nélkül.

jеле: 

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmzás (mivel az ajtó létezhet zár nélkül).

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmzás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



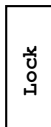
```
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
    :
```

Elsőnek készül a **Lock** osztály.

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



Elsőnek készül a **Lock** osztály.  
Majd készül a **Door** osztály.



# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



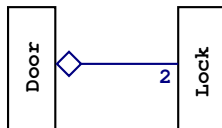
```
class Door:
    :
    def __init__(self):
        self.lck1=Lock('closed')
        self.lck2=Lock('closed')
        self.st='Door closed'
    def status(self):
        self.lck1.status()
        self.lck2.status()
        print(self.st)
    def open(self):
        self.lck1.open()
        self.lck2.open()
        self.st='Door open'
    def close(self):
        self.lck1.close()
        self.lck2.close()
        self.st='Door closed'
    :
```

Elsőnek elkészül a **Lock** osztály.  
Majd elkészül a **Door** osztály.

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.

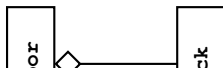
Majd elkészül a **Door** osztály.

Az ajtó két zárat tartalmaz.

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



```
class Door:
    :
    def __init__(self):
        self.lck1=Lock('closed')
        self.lck2=Lock('closed')
        self.st='Door closed'
    def status(self):
        self.lck1.status()
        self.lck2.status()
        print(self.st)
    def open(self):
        self.lck1.open()
        self.lck2.open()
        self.st='Door open'
    def close(self):
        self.lck1.close()
        self.lck2.close()
        self.st='Door closed'
    :
```

Elsőnek elkészül a **Lock** osztály.

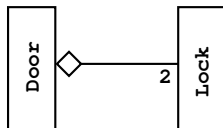
Majd elkészül a **Door** osztály.

Az ajtó két zárat tartalmaz.

# Python OOP, tárolmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tárolmazás (mivel az ajtó létezik zár nélkül).



Elsőnek elkészül a **Lock** osztály.

Majd elkészül a **Door** osztály.

Az ajtó két zárat tartalmaz.

Futtassuk!

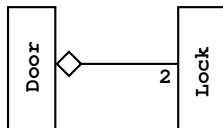
```
⋮
door=Door()
door.status()
door.open()
door.status()
door.close()
door.status()
```

Képernyő

# Python OOP, tárolás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tárolás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.

Majd elkészül a **Door** osztály.

Az ajtó két zárat tartalmaz.

Futtassuk!

```
⋮  
door=Door()  
door.status()  
door.open()  
door.status()  
door.close()  
door.status()
```

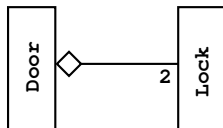
Lefut a **Door** konstruktor, ami létrehoz 2 db **Lock** objektumot.

Képernyő

# Python OOP, tárolás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tárolás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.

Majd elkészül a **Door** osztály.

Az ajtó két zárat tartalmaz.

Futtassuk!

```
:\n\ndoor=Door()\ndoor.status()\ndoor.open()\ndoor.status()\ndoor.close()\ndoor.status()
```

Lefut a **Door** konstruktor, ami létrehoz 2 db **Lock** objektumot. Kiírja a **zárak** és az **ajtó** státuszát.

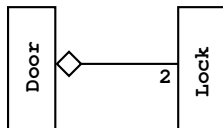
Képernyő

```
closed  
closed  
Door closed
```

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.

Majd elkészül a **Door** osztály.

Az ajtó két zárat tartalmaz.

Futtassuk!

```
:\n\ndoor=Door()\ndoor.status()\ndoor.open()\ndoor.status()\ndoor.close()\ndoor.status()
```

Lefut a **Door** konstruktor, ami létrehoz **2 db Lock** objektumot. Kiírja a **zárak** és az **ajtó** státuszát. Kinyitja az **ajtót** úgy, hogy kinyitja a **zárakat**.

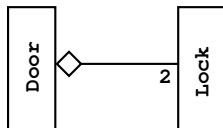
Képernyő

```
closed
closed
Door closed
```

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.

Majd elkészül a **Door** osztály.

Az ajtó két zárat tartalmaz.

Futtassuk!

```
:\n\ndoor=Door()\ndoor.status()\ndoor.open()\ndoor.status()\ndoor.close()\ndoor.status()
```

Lefut a **Door** konstruktor, ami létrehoz 2 db **Lock** objektumot. Kiírja a **zárak** és az **ajtó** státuszát. Kinyitja az **ajtót** úgy, hogy kinyitja a **zárakat**. Kiírja a **zárak** és az **ajtó** státuszát.

Képernyő

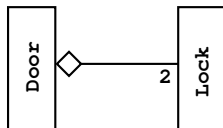
```
closed
closed
Door closed
open
open
Door open
```



# Python OOP, tárolás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tárolás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.  
Majd elkészül a **Door** osztály.  
Az ajtó két zárat tartalmaz.

Futtassuk!

```
:\n\ndoor=Door()\ndoor.status()\ndoor.open()\ndoor.status()\ndoor.close()\ndoor.status()
```

Lefut a **Door** konstruktor, ami létrehoz 2 db **Lock** objektumot. Kiírja a **zárak** és az **ajtó** státuszát. Kinyitja az **ajtót** úgy, hogy kinyitja a **zárakat**. Kiírja a **zárak** és az **ajtó** státuszát. Bezárja az **ajtót** úgy, hogy bezárja a **zárakat**.

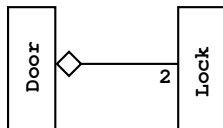
Képernyő

```
closed
closed
Door closed
open
open
Door open
```

# Python OOP, tartalmazás reláció

A példa egy ajtó két zárral.

Ez egy gyenge tartalmazás (mivel az ajtó létezhet zár nélkül).



Elsőnek elkészül a **Lock** osztály.  
Majd elkészül a **Door** osztály.  
Az ajtó két zárat tartalmaz.

Futtassuk!

```
:\n\ndoor=Door()\ndoor.status()\ndoor.open()\ndoor.status()\ndoor.close()\ndoor.status()
```

Lefut a **Door** konstruktor, ami létrehoz 2 db **Lock** objektumot. Kiírja a **zárak** és az **ajtó** státuszát. Kinyitja az **ajtót** úgy, hogy kinyitja a **zárakat**. Kiírja a **zárak** és az **ajtó** státuszát. Bezárja az **ajtót** úgy, hogy bezárja a **zárakat**. Kiírja a **zárak** és az **ajtó** státuszát.

Képernyő

```
closed
closed
Door closed
open
open
Door open
closed
closed
Door closed
```

# Python OOP, tartalmazás reláció

A forrás mégegyszer:

# Python OOP, tárolmazás reláció

A forrás mégegyszer:

## 1. Lock osztály

```
#!/usr/bin/python3
class Lock:
    def __init__(self, st):
        self.st=st
    def status(self):
        print(self.st)
    def open(self):
        self.st='open'
    def close(self):
        self.st='closed'
    :
```

# Python OOP, tartalmazás reláció

## 2. Door osztály:

```
class Door:
    def __init__(self):
        self.lck1=Lock('closed')
        self.lck2=Lock('closed')
        self.st='Door closed'
    def status(self):
        self.lck1.status()
        self.lck2.status()
        print(self.st)
    def open(self):
        self.lck1.open()
        self.lck2.open()
        self.st='Door open'
    def close(self):
        self.lck1.close()
        self.lck2.close()
        self.st='Door closed'
```

# Python OOP, tartalmazás reláció

3. a program:

```
        :  
door=Door()  
door.status()  
door.open()  
door.status()  
door.close()  
door.status()
```

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.



# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

Igen tudja, de közel sem  
úgy, mint pl. a C++!!

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name=name
    def talk(self):
        pass
class Cat (Animal):
    def talk(self):
        print('Nyau')
class Dog (Animal):
    def talk(self):
        print('Vau')
c=Cat('Cirmi')
d=Dog('Bodri')
c.talk()
d.talk()
```

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name=name
    def talk(self):
        pass
class Cat(Animal):
    def talk(self):
        print('Nyau')
class Dog(Animal):
    def talk(self):
        print('Vau')
c=Cat('Cirmi')
d=Dog('Bodri')
c.talk()
d.talk()
```

Az osztályok.

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name=name
    def talk(self):
        pass
class Cat (Animal):
    def talk(self):
        print ('Nyau')
class Dog (Animal):
    def talk(self):
        print ('Vau')
c=Cat ('Cirmi')
d=Dog ('Bodri')
c.talk()
d.talk()
```

Az osztályok.

A **talk** függvények

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name=name
    def talk(self):
        pass
class Cat(Animal):
    def talk(self):
        print('Nyau')
class Dog(Animal):
    def talk(self):
        print('Vau')
c=Cat('Cirmi')
d=Dog('Bodri')
c.talk()
d.talk()
```

Az osztályok.

A **talk** függvények

Cat objektum létrehozása.

Képernyő

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name = name
    def talk(self):
        pass
class Cat (Animal):
    def talk(self):
        print ('Nyau')
class Dog (Animal):
    def talk(self):
        print ('Vau')
c=Cat ('Cirmi')
d=Dog ('Bodri')
c.talk()
d.talk()
```

Az osztályok.

A **talk** függvények

**Cat** objektum létrehozása.

**Dog** objektum létrehozása.

Képernyő

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name=name
    def talk(self):
        pass
class Cat(Animal):
    def talk(self):
        print('Nyau')
class Dog(Animal):
    def talk(self):
        print('Vau')
c=Cat('Cirmi')
d=Dog('Bodri')
c.talk()
d.talk()
```

Az osztályok.

A **talk** függvények

**Cat** objektum létrehozása.

**Dog** objektum létrehozása.

A Cat beszél.

Képernyő

Nyau



# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name=name
    def talk(self):
        pass
class Cat(Animal):
    def talk(self):
        print('Nyau')
class Dog(Animal):
    def talk(self):
        print('Vau')
c=Cat('Cirmi')
d=Dog('Bodri')
c.talk()
d.talk()
```

Az osztályok.

A **talk** függvények

**Cat** objektum létrehozása.

**Dog** objektum létrehozása.

A **Cat** beszél.

A **Dog** beszél.

Képernyő

Nyau

Vau

# Python OOP, polimorfizmus

A polimorfizmus az objektum orientált programozás harmadik tulajdonsága. Magyarul többalakúság.

A python ismeri a polimorfizmust.

A példa az állatok hangja:

```
#!/usr/bin/python3
class Animal:
    def __init__(self, name):
        self.name=name
    def talk(self):
        pass
class Cat (Animal):
    def talk(self):
        print('Nyau')
class Dog (Animal):
    def talk(self):
        print('Vau')
c=Cat('Cirmi')
d=Dog('Bodri')
c.talk()
d.talk()
```

Az osztályok.

A **talk** függvények

Cat objektum létrehozása.

Dog objektum létrehozása.

A Cat beszél.

A Dog beszél.

Képernyő

Nyau

Vau

# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

Pont példa:

# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

Pont példa:

```
#!/usr/bin/python3

class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)

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

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

# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

Pont példa:

```
#!/usr/bin/python3

class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)

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

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

A Point osztály.

# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

Pont példa:

```
#!/usr/bin/python3

class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)

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

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

A **Point** osztály.  
Csak a kultúrált  
nyomtatásért.

# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

Pont példa:

```
#!/usr/bin/python3

class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def __str__(self):
        return "({0},{1})".format(self.x,self.y)
    def __add__(self,other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x,y)

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

A **Point** osztály.

Csak a kultúrált  
nyomtatásért.

Az összeadás operátor  
átdefiniálása a **Point**  
osztály objektumaira.



# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

Pont példa:

```
#!/usr/bin/python3

class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def __str__(self):
        return "({0},{1})".format(self.x,self.y)
    def __add__(self,other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x,y)

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

A **Point** osztály.

Csak a kultúrált  
nyomtatásért.

Az összeadás operátor  
átdefiniálása a **Point**  
osztály objektumaira.

A használata.

Képernyő

(4,6)

# Python OOP, operátor overloading

Az operátor overloading egy adott osztályra újradefiniálja a "megszokott" operátorokat.

Pont példa:

```
#!/usr/bin/python3

class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y
    def __str__(self):
        return "({0},{1})".format(self.x,self.y)
    def __add__(self,other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x,y)

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

A **Point** osztály.

Csak a kultúrált  
nyomtatásért.

Az összeadás operátor  
átdefiniálása a **Point**  
osztály objektumaira.

A használata.

Képernyő

(4,6)

# Python OOP, operátor overloading

Aritmetikai operátorok:

# Python OOP, operátor overloading

Aritmetikai operátorok:

+	<code>__add__(self, other)</code>	<code>other</code> a második operandus.
-	<code>__sub__(self, other)</code>	
*	<code>__mul__(self, other)</code>	
/	<code>__truediv__(self, other)</code>	
//	<code>__floordiv__(self, other)</code>	
%	<code>__mod__(self, other)</code>	
divmod	<code>__divmod__(self, other)</code>	
**	<code>__pow__(self, other)</code>	
+	<code>__pos__(self)</code>	unáris operátor
-	<code>__neg__(self)</code>	

# Python OOP, operátor overloading

Aritmetikai operátorok:

+	<code>__add__(self, other)</code>	<code>other</code> a második operandus.
-	<code>__sub__(self, other)</code>	
*	<code>__mul__(self, other)</code>	
/	<code>__truediv__(self, other)</code>	
//	<code>__floordiv__(self, other)</code>	
%	<code>__mod__(self, other)</code>	
divmod	<code>__divmod__(self, other)</code>	
**	<code>__pow__(self, other)</code>	
+	<code>__pos__(self)</code>	unáris operátor
-	<code>__neg__(self)</code>	

Bit operátorok:

# Python OOP, operátor overloading

## Aritmetikai operátorok:

+	<code>__add__(self, other)</code>	<code>other</code> a második operandus.
-	<code>__sub__(self, other)</code>	
*	<code>__mul__(self, other)</code>	
/	<code>__truediv__(self, other)</code>	
//	<code>__floordiv__(self, other)</code>	
%	<code>__mod__(self, other)</code>	
divmod	<code>__divmod__(self, other)</code>	
**	<code>__pow__(self, other)</code>	
+	<code>__pos__(self)</code>	unáris operátor
-	<code>__neg__(self)</code>	

## Bit operátorok:

<<	<code>__lshift__(self, other)</code>	
>>	<code>__rshift__(self, other)</code>	
&	<code>__and__(self, other)</code>	
	<code>__or__(self, other)</code>	
^	<code>__xor__(self, other)</code>	
~	<code>__invert__(self)</code>	unáris operátor

# Python OOP, operátor overloading

Relációs operátorok:

# Python OOP, operátor overloading

Relációs operátorok:

<	<code>__lt__(self, other)</code>
<=	<code>__le__(self, other)</code>
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>
>=	<code>__ge__(self, other)</code>
>	<code>__gt__(self, other)</code>



# Python OOP, operátor overloading

Relációs operátorok:

<	<code>__lt__(self, other)</code>
<=	<code>__le__(self, other)</code>
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>
>=	<code>__ge__(self, other)</code>
>	<code>__gt__(self, other)</code>

Csak a legfontosabb operátorokat ismertettük, számos további van.

# Python OOP, operátor overloading

Relációs operátorok:

<	<code>__lt__(self, other)</code>
<=	<code>__le__(self, other)</code>
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>
>=	<code>__ge__(self, other)</code>
>	<code>__gt__(self, other)</code>