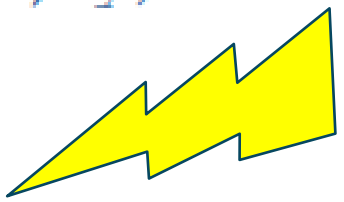
PYTHON ALAPOK OBJEKTUM ORIENTÁLTSÁG

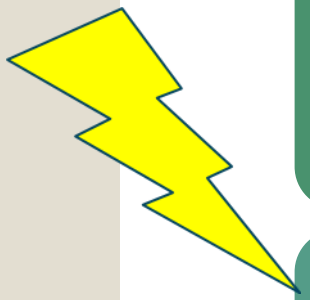
Dr. Hajnal Éva

Osztály definíció

```
class Pont:
    """A Pont osztály (x, y) koordinátáinak reprezentálására és␣
↪manipulálására. """

    def __init__(self):
        """ Egy új, origóban álló pont létrehozása. """
        self.x = 0
        self.y = 0
```



Szabályok



- Osztály helye nem rögzített, de ált. az import után
- Szintektika ugyanaz, mint az összetett utasításnál
- __init__ inicializáló metódus-konstruktor
- Self: saját objektumpéldány referenciája

Példányosítás

```
p = Pont() # A Pont osztály egy objektumának létrehozása (példányosítás)
q = Pont() # Egy második Pont objektum készítése

# Minden Pont objektum saját x és y attribútumokkal rendelkezik
print(p.x, p.y, q.x, q.y)
```

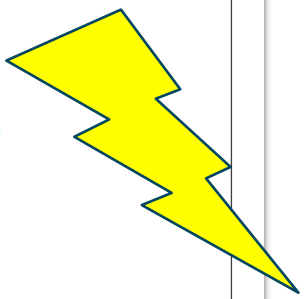
Attributumok

```
p.x = 3
p.y = 4
```

Pont operátorral megadhatók

Ugyanígy módosíthatók

```python
class Pont:
    """A Pont osztály (x, y) koordinátáinak reprezentálására és
↪manipulálására. """

    def __init__(self, x=0, y=0):
        """ Egy új, (x, y) koordinátán álló pont készítése. """
        self.x = x
        self.y = y

# További, osztályon kívül álló utasítások
```



Paraméterezett konstruktor

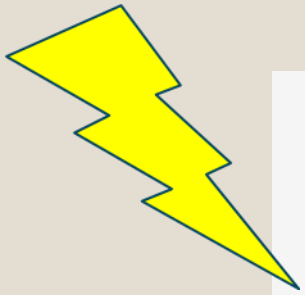
- Dokumentációs string
- +paraméter
- Alapértelmezett értékkel
- Nincs overloading

Metódusok

A függvénydefiníció szabályai szerint

Metódusok listázása dir() függvénnyel

```
# public member function
  def displayPublicMembers(self):

      # accessing public data members
      print("Public Data Member:", self.var1)
```



```
obj = Geek("R2J", 1706256, "Information Technology")
print("")
print(dir(obj))
```

Hozzáférés módosítók

Mivel script nyelv, ezért a hozzáférés módosítók rendszere egyszerűbb, mint a C#-ban

Továbbra is fontos az egységbe zárás

ENCAPSULATION

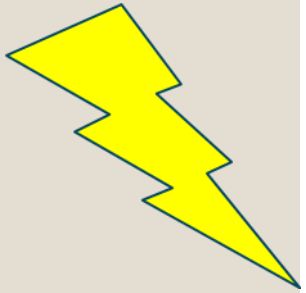
- **Private**
- Protected
- **Public**

```python
class Geek:

    # constructor
    def __init__(self, name, age):

        # public data members
        self.geekName = name
        self.geekAge = age

    # public member function
    def displayAge(self):

        # accessing public data member
        print("Age: ", self.geekAge)
```

Publikus

- Egy osztály publikus tagjai könnyen elérhetők a program bármely részéből.
- Egy osztály összes adattagja és tagfüggvénye alapértelmezés szerint nyilvános.

Private



```
class Base:

    # Declaring public method
    def fun(self):
        print("Public method")

    # Declaring private method
    def __fun(self):
        print("Private method")
```

Bármely osztály belső funkcionalitásának és belső adatainak elrejtésére szolgál a külvilág elől.

A privát metódusok azok a metódusok, amelyeket sem az osztályon kívül, sem semmilyen alaposztályban nem szabad elérni.

A Pythonban nem léteznek olyan privát metódusok, amelyekhez csak egy osztályon belül lehet hozzáférni.

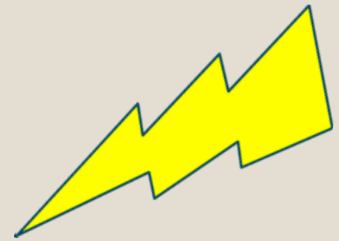
A privát metódus meghatározásához a tag nevét dupla aláhúzásjellel „__” rögzítjük.

Protected

- Egy osztály protected tagjai csak azon az osztályon és annak alosztályán belül érhetők el.
- A védett mező vagy metódus megvalósításához a fejlesztő egy speciális konvenciót követ, többnyire előtag hozzáadásával a változó vagy függvény nevéhez. Népszerű, hogy egyetlen aláhúzásjelet „_” használnak az osztály védett adattagjának vagy metódusának leírására.
- Ne feledje, hogy a python interpreter nem kezeli védett adatként, mint más nyelveket, csak a programozók számára jelöli, mivel ők egyszerű névvel próbálnák elérni, ahelyett, hogy a megfelelő előtaggal hívnák meg.

```
class Student:

    # protected data members
    _name = None
    _roll = None
    _branch = None

    # constructor
    def __init__(self, name, roll, branch):
        self._name = name
        self._roll = roll
        self._branch = branch

    # protected member function
    def _displayRollAndBranch(self):

        # accessing protected data members
        print("Roll:", self._roll)
        print("Branch:", self._branch)

# derived class
class Geek(Student):

    # constructor
    def __init__(self, name, roll, branch):
        Student.__init__(self, name, roll, branch)

    # public member function
    def displayDetails(self):

            # accessing protected data members of super class
        print("Name:", self._name)
```

Név zavarás- name mangling



A Python egy varázspálcát biztosít, amellyel az osztályon kívül is lehet privát metódusokat hívni, ezt névzavarnak nevezik.

Ez azt jelenti, hogy a __geek formátumú bármely azonosító (legalább két bevezető aláhúzás vagy legfeljebb egy záró aláhúzás) lecserélődik a _classname__geek-re,

ahol az osztálynév az aktuális osztálynév, a bevezető aláhúzás(ok) nélkül.

Ha lehet, ne alkalmazzuk!

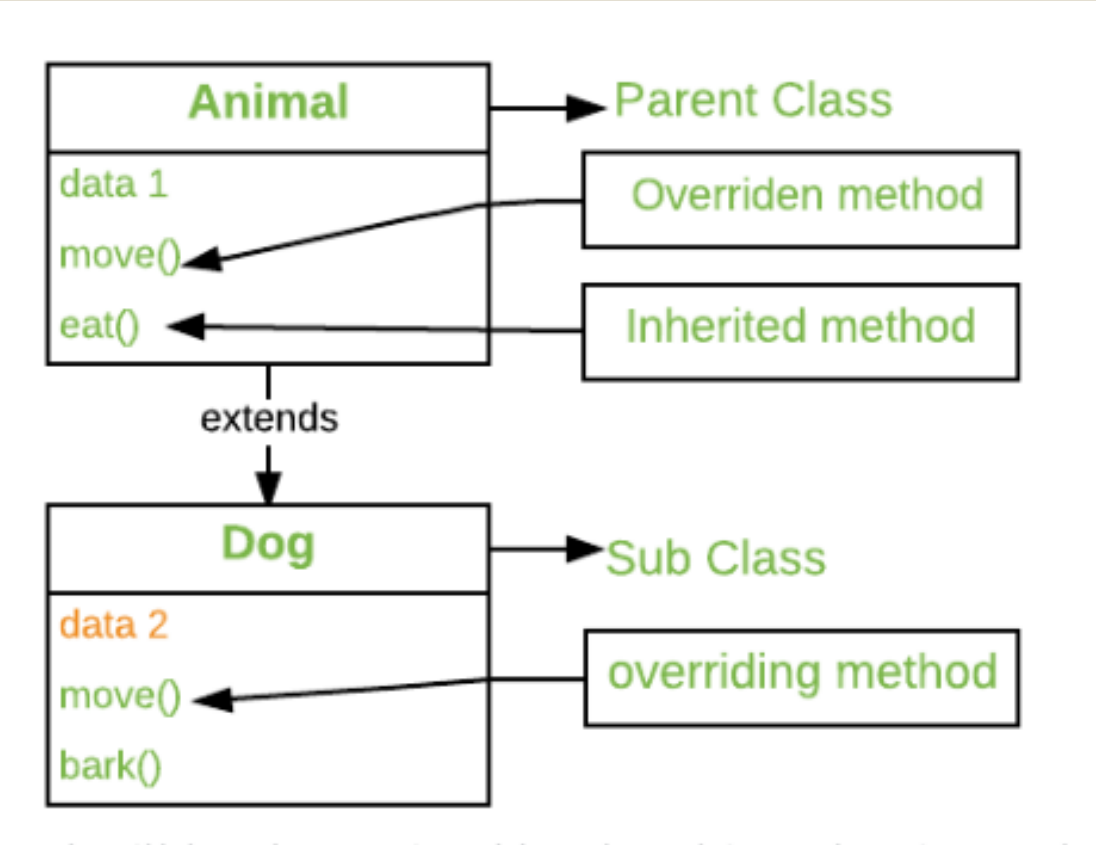
```
# Driver's code
obj = Base()
# Calling the private member
# through name mangling
obj._Base__fun()
```

```
# derived class
class Sub(Super):

    # constructor
    def __init__(self, var1, var2, var3):
        Super.__init__(self, var1, var2, var3)

    # public member function
    def accessProtectedMembers(self):

        # accessing protected member functions of super class
        self._displayProtectedMembers()
```

Öröklés

- Osztály és származtatott osztály létrehozása
- `class leszármazott_név(ős osztály_név):`
- Ősmetódus hívása – Super kulcsszóval

Overriding


Animal
Parent Class
data 1
Overriden method
move()
eat()
Inherited method
extends
Dog
Sub Class
data 2
overriding method
move()
bark()

OVERLOADING

Alapértelmezetten nincs

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
```

```python
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height
```

Interfész, absztrakt osztály

- Nyelvi implementációja alapértelmezetten nincs
- Ha akarjuk, akkor az abc (abstract base class) szabványos könyvtáron keresztül

Kivételkezelés

A **try** blokk lehetővé teszi, hogy tesztelje a kódblokkot hibákat keresve.

Az **except** blokk lehetővé teszi a hiba kezelését.

Az **else** blokk lehetővé teszi a kód futtatását, ha nincs hiba.

A **finally** blokk lehetővé teszi a kód végrehajtását, függetlenül a try-except blokkok eredményétől.

Kivételkezelés példa

```
try:
  f = open("demofile.txt")
  try:
    f.write("Lorum Ipsum")
  except:
    print("Something went wrong when writing to the file")
  finally:
    f.close()
except:
  print("Something went wrong when opening the file")
```

Kivétel dobása- raise

```
x = -1

if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

```
x = "hello"

if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

Összefoglalás

- Objektumorientáltság
- Osztály, objektum
- Tagok, metódusok
- Konstruktor
- Hozzáférés módosítók
- Öröklődés
- Kivételkezelés

- Feladat: Véletlenszerűen elhelyezett pontok köré írt kör paramétereinek kiszámítása és az eredmény megjelenítése objektumorientáltan.