Python

5. gyakorlat

Strohmayer Ádám

Agenda

- OOP alapok
- Kivételkezelés

Mi a baj az alábbi kóddal?

```
>>> john = ("John Smith", 34, "Computer Science", "cashier")
>>> blake = ("Blake Smith", 21, None, None)
>>> jane = ("Jane Smith", 18, None, "student")
>>> anonymous = (None, None, None, "advisor")
>>>
```

Objektumelvű programozás

Világot modellezzük **objektumok alapján**

- **Osztály**: "tervrajz" egy objektum számára
- **Példány**: osztály megvalósítása, memóriában tárolt valós érték

Osztályok készítése: **class Animal: ...**

Objektumorientált programozásnak 4 alapelve van!

Konstruktorok, destruktorok

- **Konstruktorok** készítik el a példányainkat!
- **Destruktorok** szabadítják fel a memóriát (ha törlünk, vagy automatikusan felszabadul a memória **(GC)**) – **ritkán írunk destruktort**!

```
>>> class Dog():
...     def __init__(self, name, breed):
...         self.name = name #attribútum!
...         self.breed = breed #attribútum!
...
...     def bark(self): #metódus!
...         print(f"{self.name}: Woof!")
...
...     def __del__(self):
...         print(f"{self.name} disappeared!")
...
```

```
>>> texas = Dog("Texas", "pug")
>>> texas.bark()
Texas: Woof!
>>> del texas
Texas disappeared!
```

```
>>> type(Dog)
<class 'type'>
```

Osztály-, példányváltozók/metódusok

- **Osztályváltozók** osztályban tárolódnak el, nem példányban! (minden példány osztozik rajta)
- **Attribútumok** mindig adott példányhoz kötött változók
- **Osztálymetódusok** futtathatók példányok nélkül (jelölése: **cls**)
- **Példánymetódusok** adott példánnyal működnek csak (jelölése: **self**)

```
class Game:
    overall_mobs_slain = 32142

    @classmethod
    def reset_stats(cls):
        cls.overall_mobs_slain = 0

Game.reset_stats()
```

Osztálymetódust @classmethod dekorátorral is jelölni kell!

Példányok, osztályok változtatása

- A példányok és osztályok **alapértelmezetten mutable** tulajdonságúak!
  - **Azaz új adattagokat tudnak fogadni példányosítás/definiálás után is!**
  - Kikerülése: **__slots__ - csak adott attribútumokat enged meg!**

```
class Cat:
    def __init__(self, name):
        self.name = name

cat = Cat("Lajos")
cat.age = 70
cat.species = "caracal"

print(cat.__dict__)
# {'name': 'Lajos', 'age':
70, 'color': 'caracal'}
```

```
class Bird:
    __slots__ = ['name', 'color']

    def __init__(self, name, color):
        self.name = name
        self.color = color

bird = Bird("Larry", "blue")
bird.age = 2  #AttributeError!
```

```python
class Storage():
    """Tároló osztály"""
    overall_stuff_count = 0 #osztályváltozó!

    def __init__(self):
        self.local_stuff = []

    def add_stuff(self, *stuff):
        if stuff:
            for item in stuff:
                self.local_stuff.append(item)
                Storage.overall_stuff_count += 1
                #hivatkozzunk az osztályra!

ny_storage = Storage()
ca_storage = Storage()

ny_storage.add_stuff("plushie", "mom's car")
ca_storage.add_stuff("gloria")

print(ny_storage.local_stuff) # ['plushie', "mom's car"]
print(Storage.overall_stuff_count) #3
```

__dict__ tárolja a tulajdonságokat!

```
Osztály:
mappingproxy({'__dict__': <attribute '__dict__' of 'Storage' objects>,
              '__doc__': 'Tároló osztály',
              '__firstlineno__': 2,
              '__init__': <function Storage.__init__ at 0x000001D220FF3E20>,
              '__module__': '__main__',
              '__static_attributes__': ('local_stuff',),
              '__weakref__': <attribute '__weakref__' of 'Storage' objects>,
              'add_stuff': <function Storage.add_stuff at 0x000001D2214B8180>,
              'overall_stuff_count': 3})
Példány:
{'local_stuff': ['gloria']}
```

Példány.attribútum formában kérjük le az adott attribútumot!

"Privát" attribútumok/ metódusok

- Közvetlenül nem elérhető változók!
  - Jelölése: _attr (**csak jelzés!**)

- **Erősebb privát attribútum:**
  - **Jelölése: __attr** (nem hívható kívülről)

  Valójában átnevezi: _osztálynév__attr

Példány __dict__-ben csak az attribútumok látszódnak!

Minden lekérdezése: **dir(példány)**

```
class Person:
    def __init__(self, name):
        self.__very_private_name = name
        self._private_name = name

    def _whisper(self):
        print("Psst!")

    def __scare(self):
        print("Peekaboo!")

jason = Person("json")
jason._whisper() # Psst!
jason.__scare() #AttributeError!
jason._Person__scare() #"Peekaboo!"
print(jason._private_name) #"json"
print(jason.__very_private_name)
#AttributeError!
print(jason._Person__very_private_name)
#"json"!
```

Dunderek

- **Osztályokban a dunderek (felül)definiálhatóak!**
  - pl. saját módon értelmezhetünk összehasonlítást, hosszt, stb.

```
class Snake:

    def __init__(self, name):
        self.name = name
        self.body = ["head","body","tail"]

    def eat(self):
        self.body.insert(1,"body")

    def __len__(self):
        return len(self.body)

    def __str__(self): #szöveges reprezentáció (felhasználóknak ált.)!
        return f"Hi, I'm {self.name}! I'm {len(self)} units long."
```

```
jeff = Snake("Jeffrey")
[jeff.eat() for _ in range(5)]
print(len(jeff)) #8
print(jeff) #__str__ alapján
```

Öröklődés

OOP céljai közé tartozik: **modularitás, rugalmasság**

Különböző elvek: KIS, SOLID, DRY…

Öröklődésnél egy osztályt egy már meglévő osztály tulajdonságaival ruházunk fel!

Csak a nyilvános adattagok öröklődnek át!

class Dog(Animal):

itt a Dog osztály **(gyerek)** megkapja az összes, nem privát tulajdonságot az Animal osztálytól **(szülő)**

Szülő metódusainak meghívása gyereken belül: super().metódus()

Felüldefiniálás, kiterjesztés

Felüldefiniálunk egy metódust, ha a gyerek osztályban teljesen átírjuk a szülő metódusát!

Kiterjesztünk egy metódust, ha a gyerek osztályban a szülő osztály metódusának viselkedését bővítjük!

Túlterhelés nincs! A legutóbb definiált metódus felül fogja írni az előzőt!

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        return f"{self.name}: ???"
```

```
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed #kiterjesztés!

    def speak(self):
        return f"{self.name}: Woof!" #felüldefiniálás!
```

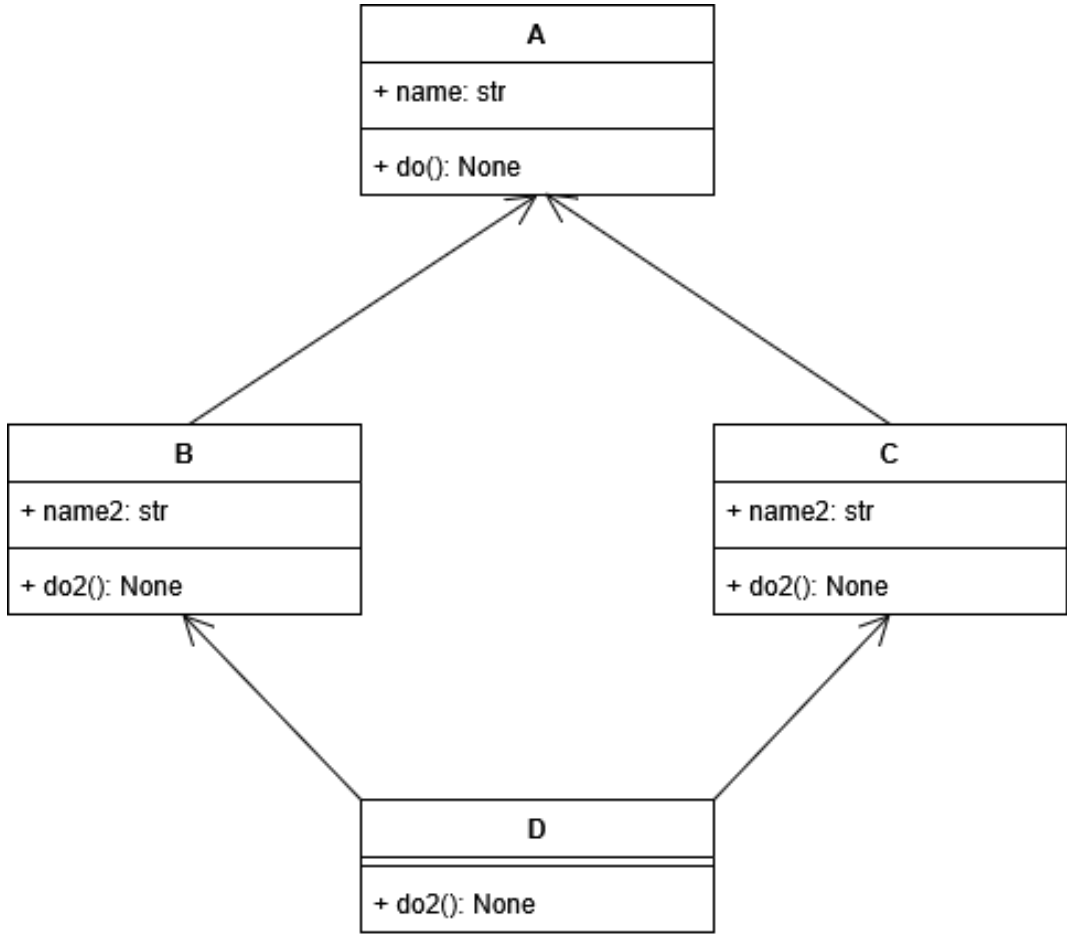
Method Resolution Order

Mi történik, ha egyszerre több osztályból származtatunk?

Feloldási probléma keletkezhet!

Ugyanolyan nevű metódusokból melyik lesz érvényes?

MRO öröklődés sorrendje szerint fogja feloldani a névkonfliktust!



MRO példa

B, C osztály elvártan örökli A osztály attribútumait, metódusait

D osztályban **számít az öröklődés sorrendje (ugyanolyan nevű metódusoknál)!**

B osztály metódusa fog kiíródni!

```
#d = D(); d.do2()
Caligula do!
B csinál
A csinál
#print(D.__mro__)
(<class '__main__.D'>, <class '__main__.B'>, <class
'__main__.C'>, <class '__main__.A'>, <class 'object'>)
```

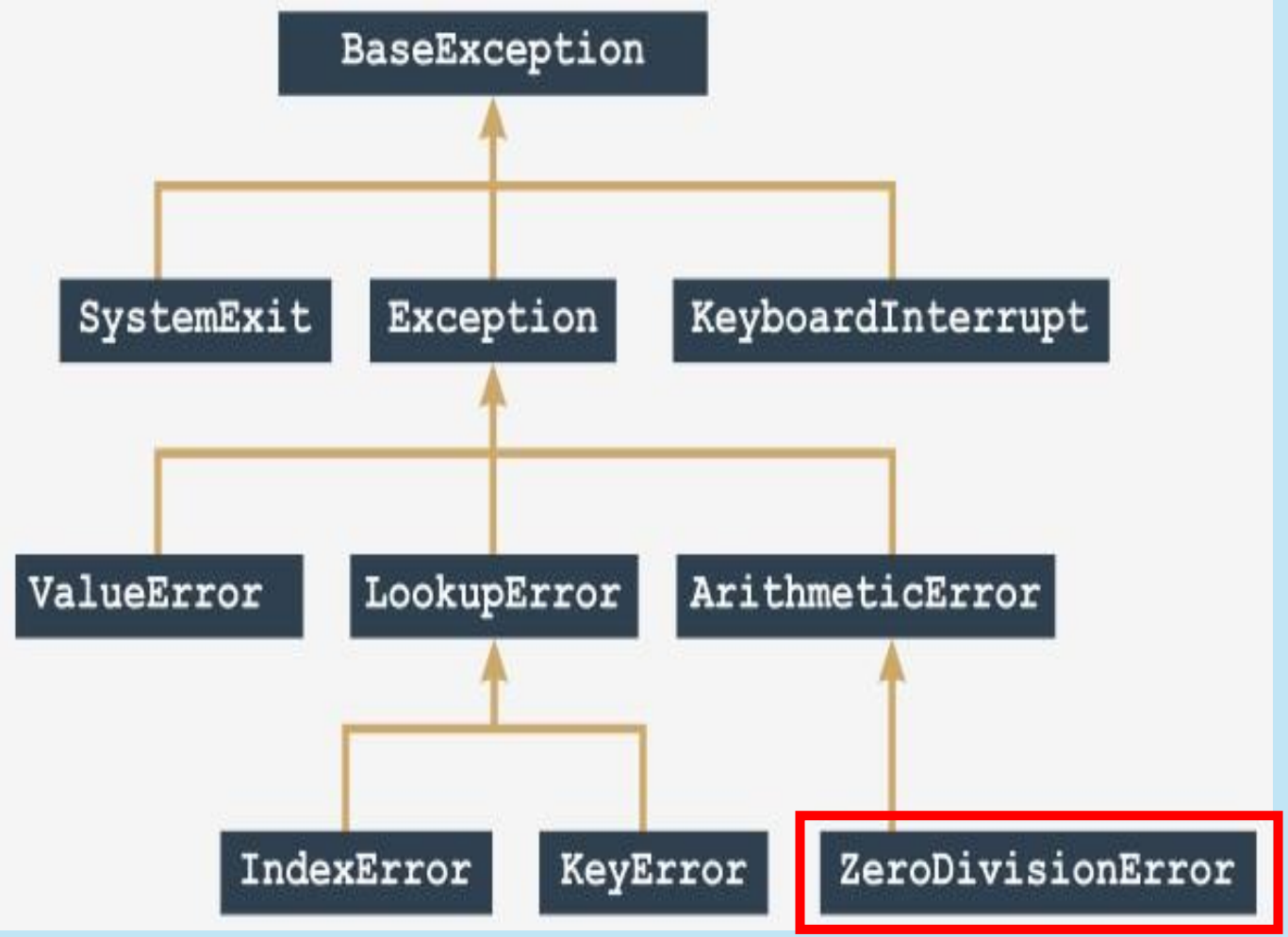
```
class A:
    name = "Caligula"
    def do(self):
        print("A csinál")

class B(A):
    def do2(self):
        print("B csinál")
        super().do()

class C(A):
    def do2(self):
        print("C csinál")
        super().do()

class D(B, C):
    def do2(self):
        print(f"{self.name} do!")
        super().do2()
```

Kivételek



Dinamikus futási hibát jeleznek!
(program futása közben lépnek fel)

(kivéve: SyntaxError, IndentationError)

Osztályonként vannak implementálva!

Fa hierarchia jellemző!

Kivételek kezelése

BaseException – legáltalánosabb kivétel

Exception – beépített kivételek osztálya

ha ebből származtatunk, saját kivételt készíthetünk!

Kivételeket try-except blokkokkal tudjuk lekezelni!

A legelső illeszkedő exceptiont fogja megtalálni **(fa hierarchia)!**

```
def divide(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        return "Hiba! Nullával nem lehet osztani!"
        #nem ér véget a program, lekezelődik a hiba!
    return result
```

Raise, assert

Raise – hibákat saját magunk **manuálisan** váltunk ki/**dobunk tovább**

pl. osztásnál előre jelezzük, hogy helytelen a paraméter

Mikor dobjuk tovább a kivételt?

Ha egy adott függvény nem bírja lekezelni kellően/nem akarunk silent failuret!

Assert – debugoláshoz, **teszteléshez** használt – **AssertionErrort dob!**

pl. invariánsok (feltételek) teljesülése ellenőrzésére teszteléskor

Nem a felhasználó hibáit szűrjük ki vele!

(python –O file.py paranccsal kikapcsolhatóak)

Saját kivételek készítése

```
class InvalidAgeError(Exception):
    """Ha nincs egy adott intervallumban a
kor"""
    def __init__(self, age, message="A kornak 6
és 120 között kell lennie!"):
        self.age = age
        self.message = message
        super().__init__(self.message)
```

```
try:
    student = Student("Jack E. Tsen", 150)
except InvalidAgeError as e:
    print(f"Error: {e}")
```

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.set_age(age)

    def set_age(self, age):
        if age < 6 or age > 120:
            raise InvalidAgeError(age)
        self.age = age

    def __str__(self):
        return f"Tanuló: {self.name}, Kor:
{self.age}"
```

finally, else

```
class LoginError(Exception):
    """Hiba helytelen felhasználónév/jelszó esetén"""
    pass
```

```
def login(username, password):
    try:
        if username != "admin":
            raise LoginError("(Helytelen felhasználónév!)")
        if password != "fradi":
            raise LoginError("(Helytelen jelszó!)")
    except LoginError as error:
        print("Hibás felhasználónév vagy jelszó!", error)
        #error üzenete kiíratva
    else:
        #ha nem kerültünk az except ágba (sikeres volt a try blokk)
        print("Sikeres bejelentkezés!")
    finally:
        #mindenképpen lefutó blokk (akár else-except ágba kerülünk előtte)
        print("Bejelentkezési próba feljegyezve.", end="\n\n")
```

Példák beépített kivételekre

- **ArithmeticError**
  Aritmetikai műveletek által okozott kivételek, mint a nullával osztás vagy egy argumentum érvénytelen tartománya.
- **AssertionError**
  Az assert utasítás sikertelen.
- **AttributeError**
  Helytelen attribútumra való hivatkozás vagy értékadás.
- **BaseException**
  A legáltalánosabb kivétel, az **except** és **except BaseException** kifejezések egyenértékűek.
- **IndexError**
  Nem található egy index a sorozatban.
- **KeyError**
  Nem létező dictionary kulcsra hivatkozunk.

- **NameError**

  Nem meghatározott lokális vagy globális név keresésekor.
- **SyntaxError**

  Az elemző szintaktikai hibát talált.
- **TypeError**

  Beépített operátornak vagy függvénynek helytelen típust adunk át.
- **ValueError**

  Argumentum hiba esetén, melyet a TypeError nem tartalmaz, ha argumentumokként érvénytelen adatokat adtunk meg.
- **ZeroDivisionError**

  Osztásnál vagy maradék (modulo) műveletnél 0 második argumentum esetében.
- **KeyboardInterrupt**

  A felhasználó megszakítja a program végrehajtását

Feladatok Canvasben!

Köszönöm a figyelmet!