Dedičnosť

Rodičia & Deti

vytvorenie rodičovskej triedy

```
## rodicovska trieda
class [RodicovskaTrieda]:
    def __init__(self, [rodicovsky_param1], [rodicovsky_param2]):
        self.[rodicovsky_atribut1] = [rodicovsky_param1]
        self.[rodicovsky_atribut2] = [rodicovsky_param2]

    def [rodicovska_metoda](self):
        return [nejaka_hodnota]

    ...
```

vytvorenie potomka

```
## trieda potomka
class [TriedaPotomka]([RodicovskaTrieda]):
    def __init__(self, [rodicovsky_param1], [rodicovsky_param2], [param_potomka]):
        self.[rodicovsky_atribut1] = [rodicovsky_param1]
        self.[rodicovsky_atribut2] = [rodicovsky_param2]
        self.[atribut_potomka] = [param_potomka]

    ## automaticky dedi vsetky metody rodica

    def [metoda_potomka](self): ## plus metody specificke pre potomka
        return [nejaka_hodnota]

    ...
```

`super()`

`super()` v `__init__`

```
## rodicovska trieda
class [RodicovskaTrieda]:
    def __init__(self, [rodicovsky_param1], [rodicovsky_param2]):
        self.[rodicovsky_atribut1] = [rodicovsky_param1]
        self.[rodicovsky_atribut2] = [rodicovsky_param2]

## trieda potomka (pouziva super() na dedenie spracovania atributov rodica)
class [TriedaPotomka]([RodicovskaTrieda]):
    def __init__(self, [rodicovsky_param1], [rodicovsky_param2], [param_potomka]):
        super().__init__([rodicovsky_param1], [rodicovsky_param2])
        self.[atribut_potomka] = [param_potomka]
```

`super()` v iných metódach

```
## rodicovska trieda
class [RodicovskaTrieda]:
    def [nazov_metody](self):
        return [rodicovska_hodnota]

## trieda potomka
class [TriedaPotomka]([RodicovskaTrieda]):
    def [nazov_metody](self):
        [rodicovsky_vysledok] = super().[nazov_metody]() ## vola metodu rodica
        ## pridanie/uprava spravania
        return [upravena_hodnota]
```

Viac informácií o metóde `super()` nájdete tu.

POLYMORFIZMUS

"Ak sa zmestí, tak to sedí"

Rovnako vyzerajúca metóda môže fungovať pre viaceré typy objektov, ak je definovaná (a/alebo priniesť rôzne výsledky).

```
## rodicovska trieda
class [RodicovskaTrieda]:
    def __init__(self, [param]):
        self.[atribut] = [param]

    def [nazov_metody](self):
        return f"{self.[atribut]} [rodicovske_spravanie]"

## trieda potomka c. 1
class [TriedaPotomka1]([RodicovskaTrieda]):
    def [nazov_metody](self):
        [rodicovsky_vysledok] = super().[nazov_metody]()
        return [rodicovsky_vysledok] + "[specificke_spravanie_potomka1]"

## trieda potomka c. 2
class [TriedaPotomka2]([RodicovskaTrieda]):
    def [nazov_metody](self):
        [rodicovsky_vysledok] = super().[nazov_metody]()
        return [rodicovsky_vysledok] + "[specificke_spravanie_potomka2]"

[instancia_rodicovskeho_objektu].[nazov_metody]()   ## toto bude fungovat
[instancia_potomka1].[nazov_metody]()   ## toto bude fungovat
[instancia_potomka2].[nazov_metody]()   ## aj toto bude fungovat
```

Duck Typing

"Ak to kváka ako kačica, je to kačica"

```
## vsimnite si, ze nizsie uvedene triedy su uplne nezavisle!
## trieda c. 1
class [Trieda1]:
    def [nazov_metody](self):
        return "[spravanie_triedy1]"

## trieda c. 2
class [Trieda2]:
    def [nazov_metody](self):
        return "[spravanie_triedy2]"

## trieda c. 3
class [Trieda3]:
    def [nazov_metody](self):
        return "[spravanie_triedy3]"

## funkcia vyuzivajuca duck typing
def [nazov_funkcie]([obj]):
    return [obj].[nazov_metody]()    ## ak objekt ma taku metodu, bude fungovat

[nazov_funkcie]([instancia_triedy1])  ## toto bude fungovat
[nazov_funkcie]([instancia_triedy2])  ## toto bude fungovat
[nazov_funkcie]([instancia_triedy3])  ## aj toto bude fungovat
```

Všeobecné rady

1. **Nie ste si istí, prečo používať `super()` mimo `__init__`, keď všetky metódy sa aj tak dedia?** Používajte `super()` v iných metódach, keď (i) rozširujete správanie rodičovskej metódy, alebo (ii) prepisujete správanie rodičovskej metódy, ale len čiastočne.