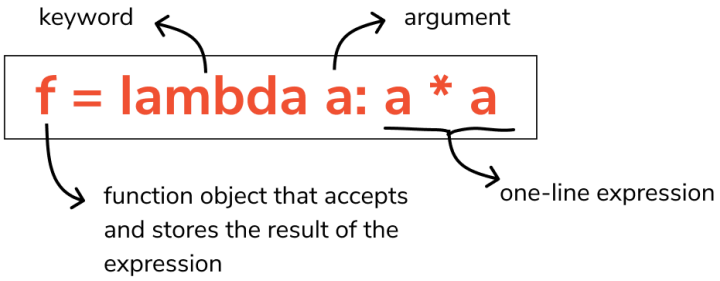
Anonymní metody
Anonymní metody neboli lambda funkce jsou krátké, bezejmenné funkce, které jsou vytvářeny přímo v místě použití. Zapisují se pomocí klíčového slova (v Pythonu `lambda`), poté parametry, dvojtečka a výraz. Jejich hlavní charakteristikou je, že nepotřebují být zvlášť pojmenovány jako běžné funkce.



Lambda funkce mají výhodu v tom, že jsou velmi stručné a udržují kód krátký a přehledný při jednoduchých operacích. Další výhodou je možnost definování přímo v místě použití bez potřeby vytvářet zbytečně pojmenované funkce.

V Pythonu jsou lambdy omezené na jeden výraz a používají se hlavně s funkcemi vyššího řádu jako map nebo filter. V Javě byly lambda výrazy přidány v Java 8 a fungují s funkčními rozhraními - mohou obsahovat více příkazů a mají přístup k proměnným z vnějšího bloku, pokud jsou efektivně finální. V C# jsou lambdy úzce spojené s delegáty a LINQ, mají flexibilnější syntaxi a plnou podporu uzávěrů včetně modifikace proměnných z vnějšího rozsahu.

Speciální magické metody
Speciální metody, označované také jako magické nebo dunder metody (z anglického "double underscore"), představují jedinečný koncept v objektově orientovaném programování v Pythonu. Tyto metody umožňují vývojářům definovat, jak se jejich vlastní třídy mají chovat v různých situacích a kontextech.

Magické metody jsou speciální funkce v Pythonu, které jsou vždy obklopeny dvojitými podtržítky na začátku a na konci jejich názvu, například `__init__`, `__str__` nebo `__add__`. Tento specifický způsob pojmenování slouží k jasnému odlišení těchto metod od běžných metod, protože mají zvláštní význam pro interpret Pythonu.

Tyto metody jsou "magické" v tom smyslu, že jsou volány automaticky Pythonem v reakci na určité operace nebo kontexty - programátor je většinou přímo nevolá. Například metoda `__add__` je volána, když použijeme operátor `+` mezi objekty, nebo metoda `__str__` je volána, když použijeme funkci `print()` na objekt.

nejzákladnejší jsou __init:__, __str__, __eq__, __len__ …..

2. `__str__` - Textová reprezentace pro uživatele

```python
class Student:
    def __init__(self, jmeno, vek):
        self.jmeno = jmeno
        self.vek = vek

    def __str__(self):
        return f"Student {self.jmeno}, {self.vek} let"

# Print automaticky volá __str__
student = Student("Jan", 18)
print(student)  # Vypíše: Student Jan, 18 let
```

Přetěžování operátorů je jeden z nejdůležitějších konceptů spojených s magickými metodami v Pythonu. Jedná se o techniku, která umožňuje používat standardní operátory Pythonu (jako +, -, *, /, ==, <, > atd.) s vlastními třídami, přičemž definujeme, co tyto operátory znamenají v kontextu našich tříd. V podstatě to můžeme upravit podle sebe co ty operátory dělají

```python
class Penize:
    def __init__(self, hodnota):
        self.hodnota = hodnota

    def __str__(self):
        return f"{self.hodnota} Kč"

    # Přetížení operátoru +
    def __add__(self, other):
        if isinstance(other, Penize):
            return Penize(self.hodnota + other.hodnota)
        return Penize(self.hodnota + other)

    # Přetížení operátoru -
    def __sub__(self, other):
        if isinstance(other, Penize):
            return Penize(self.hodnota - other.hodnota)
        return Penize(self.hodnota - other)

    # Přetížení operátoru ==
    def __eq__(self, other):
        if isinstance(other, Penize):
            return self.hodnota == other.hodnota
        return self.hodnota == other


mzda = Penize(25000)
bonus = Penize(5000)

# Sčítání peněz
celkem = mzda + bonus  # Výsledek: 30000 Kč
prilepek = mzda + 1000  # Výsledek: 26000 Kč

# Odčítání
po_nakupu = celkem - 5000  # Výsledek: 25000 Kč

# Porovnávání
if mzda == 25000:
    print("Správná mzda!")

if mzda + bonus == celkem:
    print("Matematika funguje!")
```

Statické metody
Statické metody jsou dalším typem speciálních metod, které se odlišují od běžných instančních metod. Statické metody patří výhradně dané třídě, ve které jsou, nikoliv jednotlivým instancím této třídy a pracují na nich nezávisle.

Instanční metoda pracuje s konkrétní instancí třídy a má přístup k jejím vlastnostem pomocí parametru self. Naproti tomu statická metoda nepřijímá parametr self a nemá přístup k atributům instance. Je totálně nezávislá na tom, zda vůbec nějaký objekt existuje. Pozná se podle dekorátoru @staticmethod v Pythonu.

```
class Kalkulacka:
    @staticmethod
    def secti(a, b):
        return a + b

    @staticmethod
    def odecti(a, b):
        return a - b


# Volání statické metody přímo z třídy
vysledek = Kalkulacka.secti(5, 3)  # Vrátí 8
```

Statické metody se využívají když potřebujeme zajistit funkci, která souvisí s třídou, ale nepoužívá její instanci, nepotřebuje přistup k atributům instance ani třídy a je užitečná sama o sobě, bez nutnosti vytvářet objekt.

Delegáti
Ukazatel na metodu je koncept, který umožňuje pracovat s metodami jako s běžnými objekty - můžete je přiřadit do proměnné, předat jako argument funkci nebo je uložit do datové struktury.

V pythonu jsou metody objekty prvního řádu, to znamená,že s nimi můzeme manipulovat stejně jak s jinými hodnotami. když přistupujeme k metodě objektu bez volání, získáme “bound method” - tato metoda se váže na konkrétní instanci.

Tato vázaná metoda si s sebou nese referenci na instanci -self , takže když ji později zaoválme, python už předem ví, se kterým objektem pracovat.

```
class Osoba:
    def __init__(self, jmeno):
        self.jmeno = jmeno

    def pozdrav(self):
        return f"Ahoj, já jsem {self.jmeno}"

    def rozlouceni(self):
        return f"Nashledanou, měj se hezky. S pozdravem {self.jmeno}"


# Vytvoření instance
jan = Osoba("Jan")

# Uložení ukazatele na metodu do proměnné
metoda_pozdravu = jan.pozdrav

# Pozdější volání metody
print(metoda_pozdravu())  # Vypíše: Ahoj, já jsem Jan
```

V C# to funguje jinak než v pythonu. Místo jednoduchého přiřazení metody do proměnné v C# se pracuje s delegáty - speciální typy, které slouží jako ukazatele na metody.

Delegát v C# je v podstatě typ, který definuje návratový typ a typy parametrů. Když vytvoříme instanci delegáta, stává se referencí na konkretní metodu se stejnou signaturou

Proces - 1. deklerace delegáta - určuje se jaké metody uložit do delegáta
2. vytvoření instance delegáta a přiřazení metody
3. volání delegáta

Delegáti v C# zajišťují -
typovou bezpečnost = compiler kontroluje správný počet parametrů i návratových typů

multicast = možnost uložit do jednoho delegáta více metod, tím pádem máme možnost zavolat všechny najednou, udržuje pořadí ve kterém byly metody přidávány

C# obsahuje 3 vestavěné generické delegáty, které umožňují práci s metodami bez nutnosti deklerace vlastních delegátů

1. Func <T,Tresult>
   používá se pro metody, které vracejí hodnotu. Poslední generický parametr vždy určuje návratový typ metody

2. Action <T>
   používá se pro metody, které nevracejí hodnotu. Generické parametry určují typy argumentů metody.

3. Predicate <T>
   speciální případ func, tkerý příjmá jeden parametr a vrací bool. Slouží například pro testování podmínek nebo filtrování.