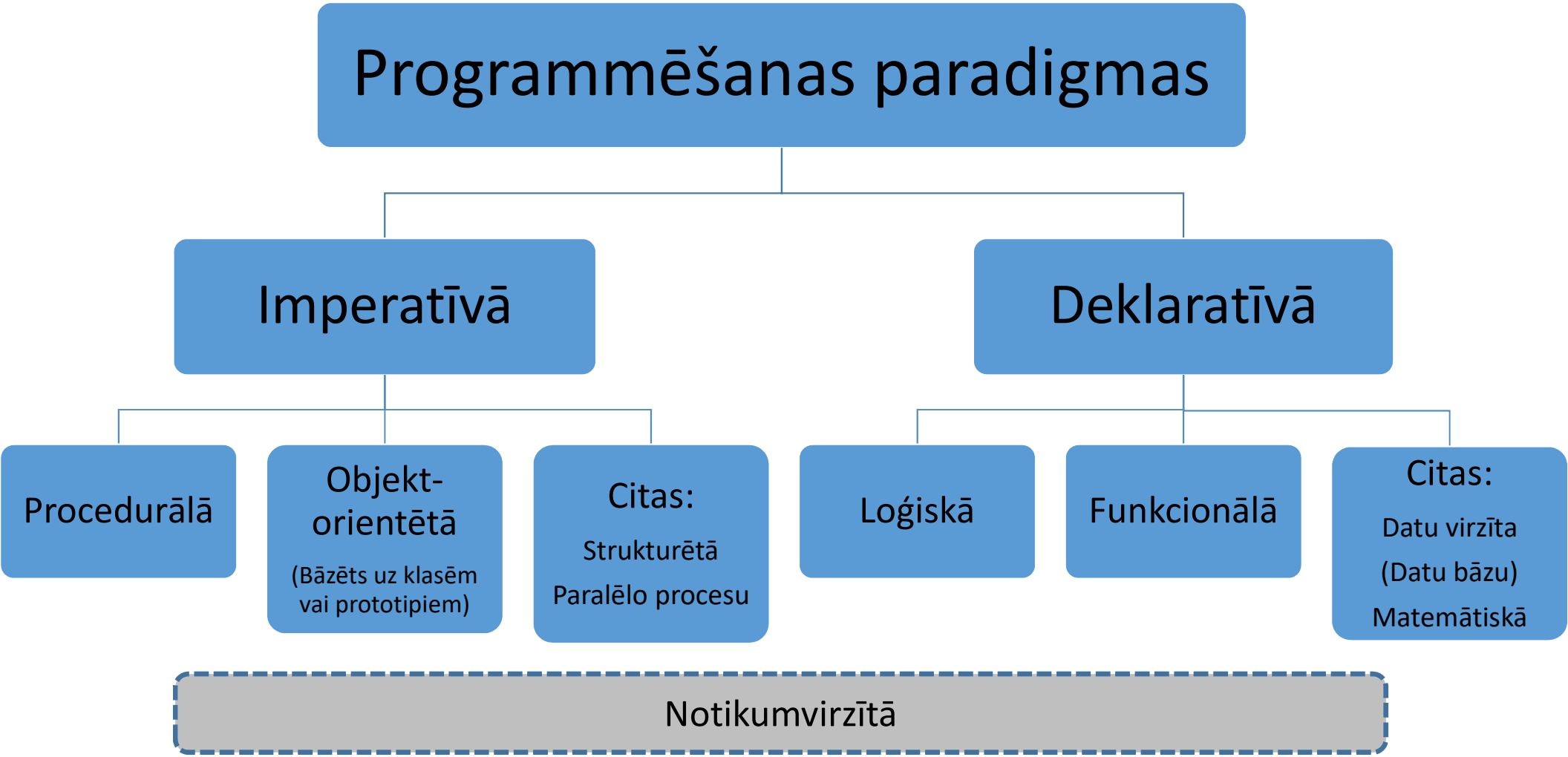
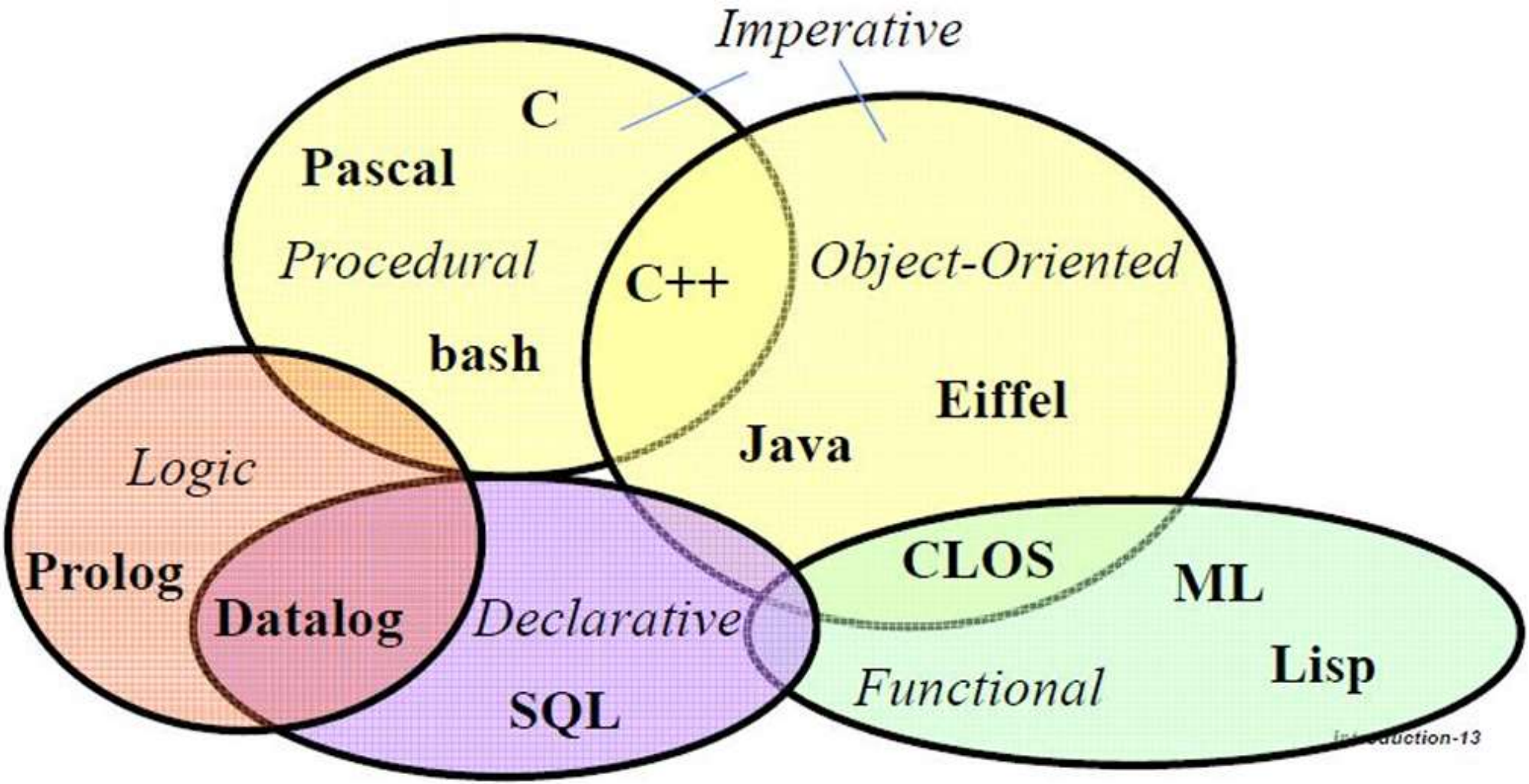
Ieskats Objektorientētā programmēšanā

Viesturs Vēzis


Programmēšanas paradigmas
Imperatīvā
Deklaratīvā
Procedurālā
Objekt-orientētā
(Bāzēts uz klasēm vai prototipiem)
Citas:
Strukturētā
Paralēlo procesu
Loģiskā
Funkcionālā
Citas:
Datu virzīta
(Datu bāzu)
Matemātiskā
Notikumvirzītā

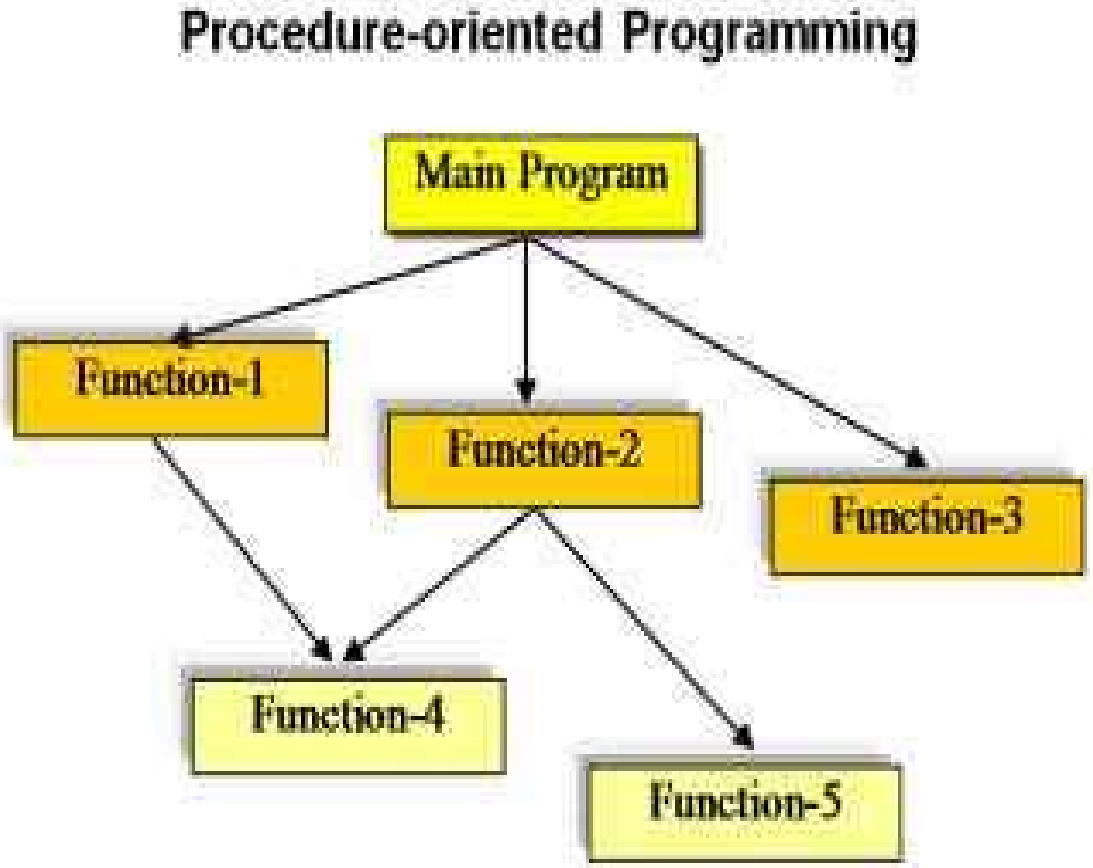
Programmēšanas valodas un programmēšanas paradigmas

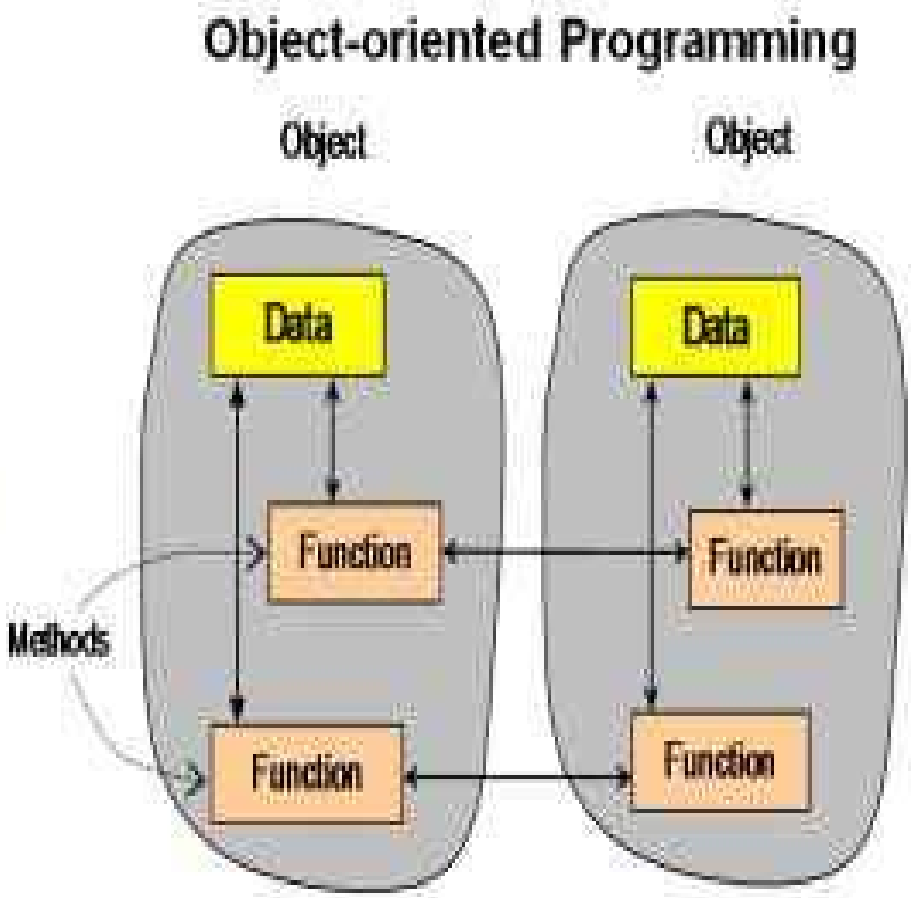


Salīdzinājums

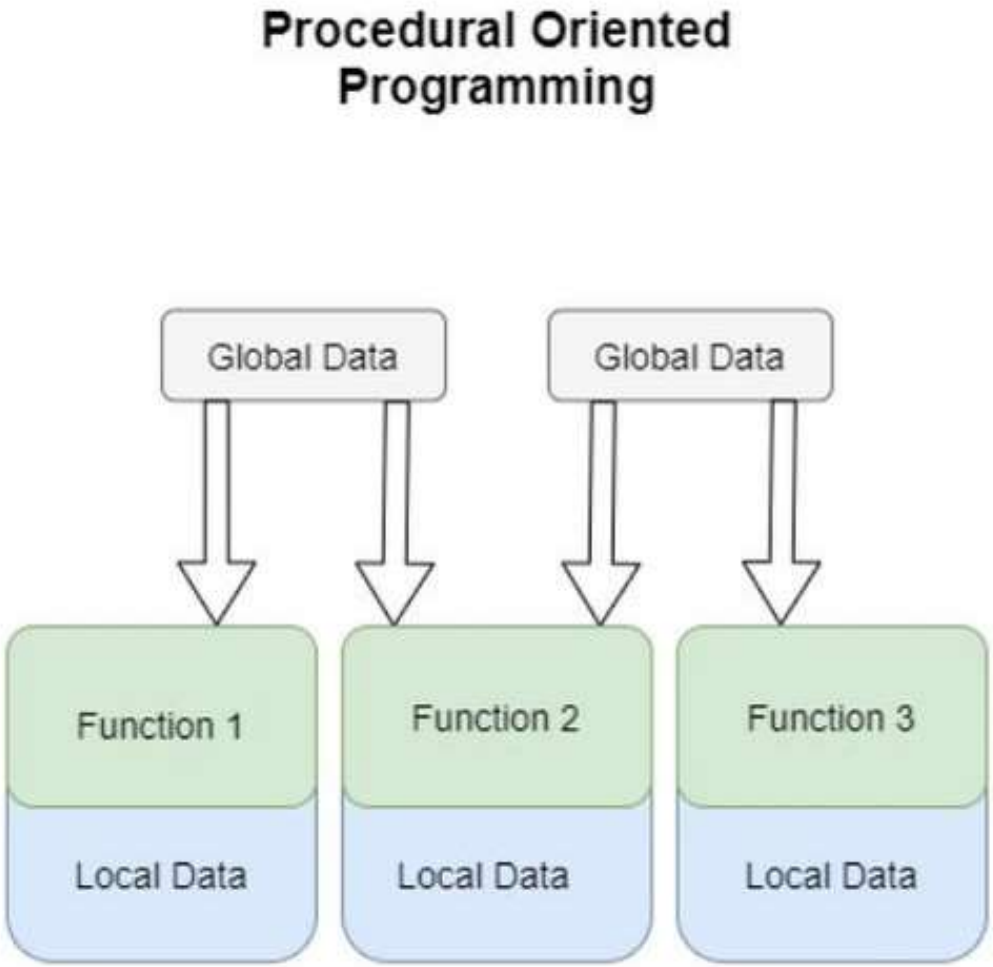
Objektorientētā programmēšana	**Procedurālā (uz procedūrām orientētā) programmēšana**
Strukturē pēc datiem	Strukturē pēc procedūrām
Programma ir sadalīta objektos	Programma ir sadalīta funkcijās (un procedūrās)
Programmas veidošanas pieeja no apakšas uz augšu	Programmas veidošanas pieeja no augšas uz leju

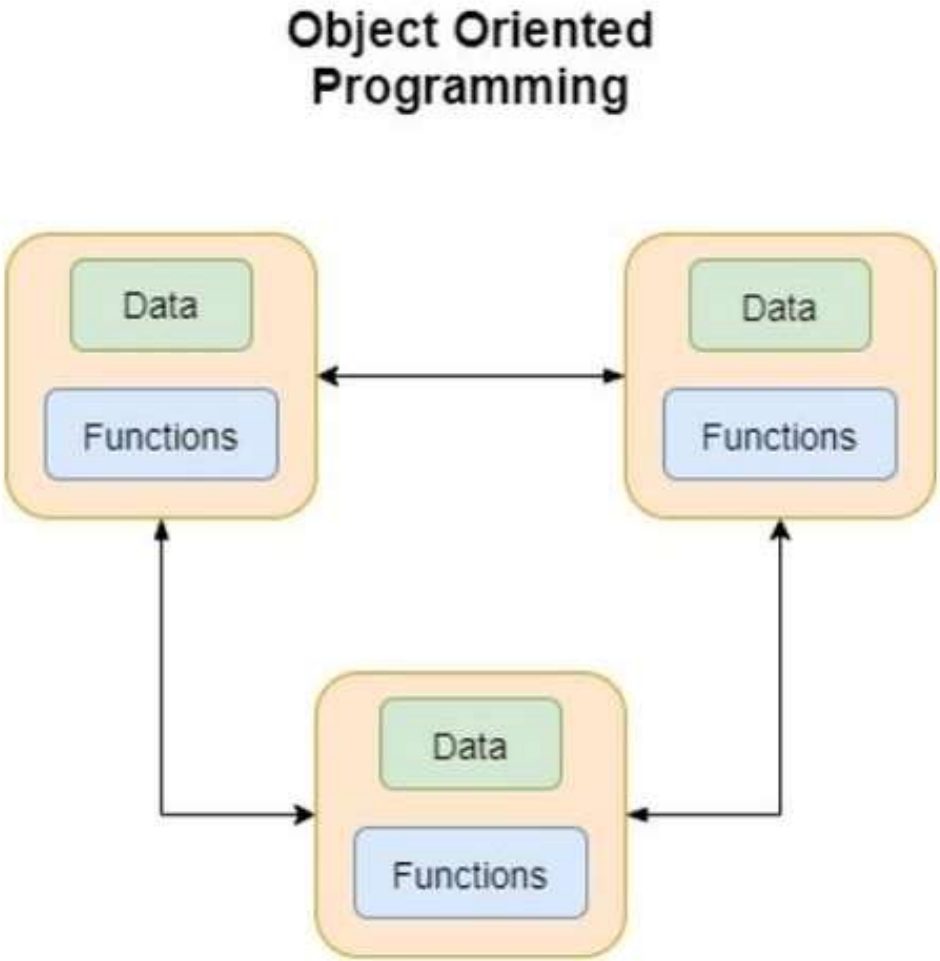
Salīdzinājums





Salīdzinājums





OPP pamatjēdzieni

- Objekts
- Klase
- Mantošana
- Polimorfisms
- Iekapsulēšana
- Abstrakcija

OPP pamatjēdzieni

KLASE

Automašīnas

OBJEKTI

AUDI

BMW

VOLVO

OPP pamatjēdzieni

- **Objekts** sastāv no datu kopuma un šiem datiem piesaistītajām darbībām.
  Datus objektā reprezentē iekšējie mainīgie jeb ***lauki***,
  bet darbības reprezentē iekšējās funkcijas (un procedūras) jeb ***metodes***.
  Objekti tiek izveidoti programmas darbības gaitā un glabājas datora atmiņā.

  EN *object*
  LV *objekts*
  prot. Nr. 567 (04.12.2020)
  LZA TK ITTEA protokoli

OPP pamatjēdzieni

- **Klase** ir apraksts, pēc kura tiek izveidoti ***objekti***.
  Šajā aprakstā tiek aprakstīti jeb deklarēti objekta iekšējie mainīgie un funkcijas.
  Šo aprakstu veido programmētājs un tas ir daļa no programmas koda.
  Ja objekts ir izveidots pēc dotā apraksta, tad saka ka tas pieder attiecīgajai klasei jeb ir attiecīgās ***klases instance***.

  EN *class*
  LV *klase*
  prot. Nr. 567 (04.12.2020), ISO/IEC/IEEE 24765:2017(E) 3.577, prot. Nr. 575 (09.04.2021)
  LZA TK ITTEA protokoli

Objekts, instance un klases instance

- **Instance** *(vesturiski – eksemplārs – līdzīgi kā krievu valodā)*

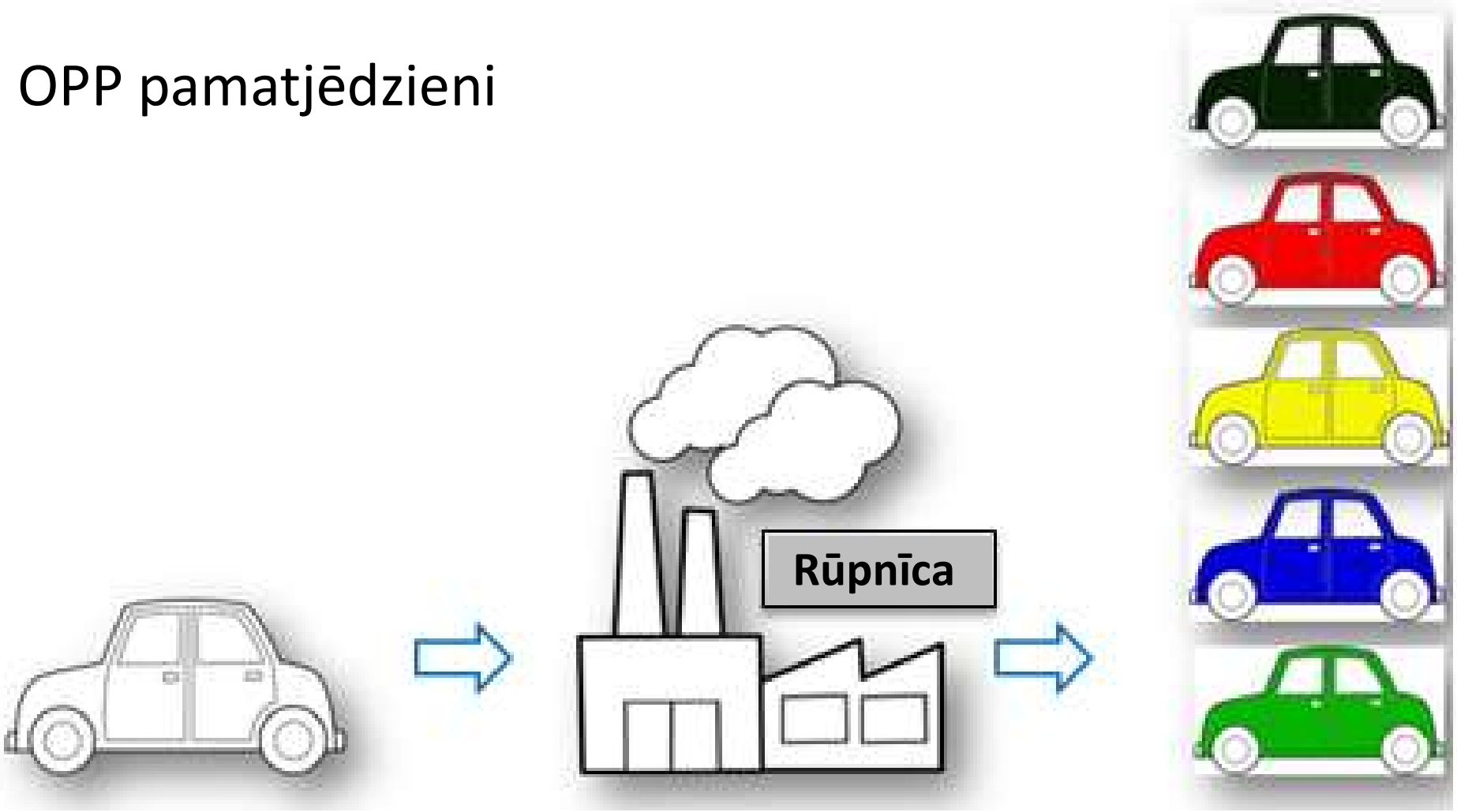
  EN instance

  LV instance

  prot. Nr. 567 (04.12.2020)
  LZA TK ITTEA protokoli

- Personām, kas neiedziļinās detaļās, ***objekts*** un ***instance*** ir viens un tas pats, bet …
- **Objekts** ir OOP pamatjēdziens
  un parasti lietojam, ja nevēlamies akcentēt objekta piederību kādai konkrētai klasei
- **Instance** parasti lietojam, ja gribam uzsvērt objekta piederību kādai konkrētai klasei
- **Instance,** *klases instance, klases nosaukums instance* uztverami kā sinonīni un lietojami atbilstoši pēc nepieciešamības, lai klausītājs labāk uztvertu stāstījumu.

OPP pamatjēdzieni



KLASE
Automašīnas modeļa apraksts

KONSTUKTORS
Automašīnu ražošana

OBJEKTI
Saražotās automašīnas

KLASE

Automašīnas modeļa apraksts

- Dati (Lauki):
  - modelis
  - gads
  - dzinēja_tips
  - dzinēja_tilpums
  - ātrumu_kārba
  - virsbūves_tips
  - krāsa
- Darbības (Metodes):
  - iedarbināt()
  - apturēt()
  - braukt()
  - paātrināt()
  - bremzēt()
  - signalizēt()

OBJEKTI

Saražotās automašīnas

- Dati (Lauki):
  - modelis = **"BMW X7"**
  - gads = **2022**
  - dzinēja_tips = **"benzīns"**
  - dzinēja_tilpums = **3.0**
  - ātrumu_kārba = **"automāts"**
  - virsbūves_tips = **"apvidus"**
  - krāsa = **"sarkana"**
- Darbības (Metodes):
  - iedarbināt()
  - apturēt()
  - braukt()
  - paātrināt()
  - bremzēt()
  - signalizēt()

- Dati (Lauki):
  - modelis = **"BMW X1"**
  - gads = **2012**
  - dzinēja_tips = **"dīzelis"**
  - dzinēja_tilpums = **2.0**
  - ātrumu_kārba = **"manuāla"**
  - virsbūves_tips = **"apvidus"**
  - krāsa = **"zila"**
- Darbības (Metodes):
  - iedarbināt()
  - apturēt()
  - braukt()
  - paātrināt()
  - bremzēt()
  - signalizēt()

Klases un klases instances (objekti)

Klasiskā pieeja OOP – lauku izveide

```
class Auto:
    modelis = ""
    gads = 2000
    krasa = ""

# 1.objekta izveide
bembis = Auto()
bembis.modelis = "BMW X7"
bembis.gads = 2022
bembis.krasa = "sarkana"
```

```
# 2.objekta izveide
opelis = Auto()
opelis.modelis = "Opel Corsa"
opelis.gads = 2001
opelis.krasa = "zila"

# ================================
print(bembis.modelis)
print(bembis.gads)              #!!!
print(bembis.krasa)
print(opelis.modelis)
print(opelis.gads)              #!!!
print(opelis.krasa)
```

```
class Auto:
    modelis = ""
    gads = 2000
    krasa = ""

# 1.objekta izveide
bembis = Auto()
bembis.modelis = "BMW X7"
bembis.gads = 2022
bembis.krasa = "sarkana"
```

```
# 2.objekta izveide
opelis = Auto()
opelis.modelis = "Opel Corsa"

opelis.krasa = "zila"

# ================================
print(bembis.modelis)
print(bembis.gads)              #!!!
print(bembis.krasa)
print(opelis.modelis)
print(opelis.gads)              #!!!
print(opelis.krasa)
```

Jauna lauka pievienošana un dzēšana (valodas *Python* īpatnība)

```
class Auto:
    modelis = ""
    gads = 2000
    krasa = ""

# 1.objekta izveide
bembis = Auto()
bembis.modelis = "BMW X7"
bembis.gads = 2022
bembis.krasa = "sarkana"

# 1.objekta apstrāde
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
bembis.gads = 2022
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
```

```
class Auto:
    modelis = ""
    gads = 2000
    krasa = ""

# 1.objekta izveide
bembis = Auto()
bembis.modelis = "BMW X7"
bembis.gads = 2022
bembis.krasa = "sarkana"

# 1.objekta apstrāde
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
bembis.gads = 2022
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
```

```
# jauna lauka pievienošana objektam

bembis.nummurs = "AA1234"
print(bembis.nummurs)
bembis.nummurs = "KRUTAIS"
print(bembis.nummurs)
```

```
class Auto:
    modelis = ""
    gads = 2000
    krasa = ""

# 1.objekta izveide
bembis = Auto()
bembis.modelis = "BMW X7"
bembis.gads = 2022
bembis.krasa = "sarkana"

# 1.objekta apstrāde
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
bembis.gads = 2022
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
```

```
# jauna lauka pievienošana objektam

bembis.nummurs = "AA1234"
print(bembis.nummurs)
bembis.nummurs = "KRUTAIS"
print(bembis.nummurs)


# lauka dzēšana

del bembis.modelis
```

```
class Auto:
    modelis = ""
    gads = 2000
    krasa = ""

# 1.objekta izveide
bembis = Auto()
bembis.modelis = "BMW X7"
bembis.gads = 2022
bembis.krasa = "sarkana"

# 1.objekta apstrāde
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
bembis.gads = 2022
print(bembis.modelis)
print(bembis.gads)
print(bembis.krasa)
```

```
# jauna lauka pievienošana objektam

bembis.nummurs = "AA1234"
print(bembis.nummurs)
bembis.nummurs = "KRUTAIS"
print(bembis.nummurs)


# lauka dzēšana

del bembis.modelis


# objekta dzēšana

del bembis
```

Tehniski valodā *Python* var arī tā

```
class Auto:
    pass

# 1.objekta izveide
bembis = Auto()
bembis.modelis = "BMW X7"
bembis.gads = 2022
bembis.krasa = "sarkana"
```

```
# 2.objekta izveide
opelis = Auto()
opelis.modelis = "Opel Corsa"
opelis.gads = 2001
opelis.krasa = "zila"

# ================================
print(bembis.modelis)
print(bembis.gads)
print(opelis.modelis)
print(opelis.gads)
opelis.gads = 2023
print(opelis.modelis)
print(opelis.gads)
```

Konstruktors un metodes
(konstruktors ir «īpaša» metode)
un
ieteicamais *lauku* izveidošanas veids
no klasiskā skata punkta

```
class Auto:
    def __init__(self):                                    # konstruktors
        self.modelis = "BMW X7"
        self.gads = 2022
        self.krasa = "sarkana"

    def izdrukat(self):                                    # metode
        print("Automašīna:", self.modelis, self.gads, self.krasa)

bembis1 = Auto()
bembis2 = Auto()
bembis1.izdrukat()
bembis2.izdrukat()
```

```
class Auto:
    def __init__(self):                                    # konstruktors
        self.modelis = "BMW X7"
        self.gads = 2022
        self.krasa = "sarkana"

    def izdrukat(self):                                    # metode
        print("Automašīna:", self.modelis, self.gads, self.krasa)

bembis1 = Auto()
bembis2 = Auto()
bembis1.izdrukat()
bembis2.izdrukat()



# Valodā Python parasti lieto self,
# bet principā te var būt jebkurš cits identifikators, piemēram, pats
```

```
class Auto:
    def __init__(self, modelis, gads, krasa):     # formālie parametri
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa

    def izdrukat(self):
        print("Automašīna:", self.modelis, self.gads, self.krasa)

bembis1 = Auto("BMW X7", 2022, "Sakana")          # argumenti
bembis2 = Auto("BMW X7", 2022, "Balta")
bembis1.izdrukat()
bembis2.izdrukat()

bembis2.krasa = "Zila"
bembis1.izdrukat()
bembis2.izdrukat()
```

```
class Auto:
    def __init__(self, modelis1, gads1, krasa1): # NEVĒLAMAIS VARIANTS!
        self.modelis = modelis1
        self.gads = gads1
        self.krasa = krasa1

    def izdrukat(self):
        print("Automašīna:", self.modelis, self.gads, self.krasa)

bembis1 = Auto("BMW X7", 2022, "Sakana")
bembis2 = Auto("BMW X7", 2022, "Balta")
bembis1.izdrukat()
bembis2.izdrukat()

bembis2.krasa = "Zila"
bembis1.izdrukat()
bembis2.izdrukat()
```

```
class Auto:
    def __init__(self, modelis, gads, krasa):                    #!
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa

    def izdrukat(self):
        print("Automašīna:", self.modelis, self.gads, self.krasa)

bembis1 = Auto("BMW X7", gads=2023, krasa="Sarkana")              #!
bembis2 = Auto(krasa="Balta", modelis="BMW X1", gads=2022)        #!
bembis1.izdrukat()
bembis2.izdrukat()
bembis2.krasa = "Zila"
bembis1.izdrukat()
bembis2.izdrukat()
```

```
class Auto:
    def __init__(self, modelis="BMW X7", gads=2022, krasa="Zaļa"):   #!
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa

    def izdrukat(self):
        print("Automašīna:", self.modelis, self.gads, self.krasa)

bembis1 = Auto(modelis="BMW X1", gads=2023, krasa="Melna")           #!
bembis2 = Auto("BMW X3", krasa="Balta")                              #!
bembis3 = Auto()                                                     #!
bembis1.izdrukat()
bembis2.izdrukat()
bembis3.izdrukat()
bembis2.krasa = "Zila"
bembis1.izdrukat()
bembis2.izdrukat()
```

```
class Auto:
    def __init__(self, modelis="BMW X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
        self.atrums = 0                                          #!

    def izdrukat(self):
        print("Automašīna:", self.modelis, self.gads,
              self.krasa, self.atrums)

    def paatrinat(self, paatrinajums=1):                         #!
        self.atrums = self.atrums + paatrinajums

bembis1 = Auto("BMW X5", gads = 2023, krasa = "Sarkana")
bembis1.izdrukat()
bembis1.paatrinat()
bembis1.izdrukat()
bembis1.paatrinat(7)
bembis1.izdrukat()
```

Klases un objekta (instances) elementi

- **Objekts** sastāv no datu kopuma un šiem datiem piesaistītajām darbībām.
  Ja nešķirojam datus un šiem datiem piesaistītās dabības, tad klasē un objektā *(instancē) to reprezentējošos vienumus vienā vārdā sauc attiecīgi* ***klases elementi*** *un* ***objekta elementi (instances elementi)***

EN class member	**EN object member**	**EN instance member**
LV klases elements	**LV objekta elements**	**LV instances elements**

prot. Nr. 605 (21.10.2022)
LZA TK ITTEA protokoli

- Taču skatoties detalizēti uz katru klases un objekta (instances) elementu gan terminoloģiski ir vērojama liela dažādība, bet vienotu sistēmu var uzkonstruēt. To esamība vai neesamība saistīta ar konkrēto programmēšanas valodu un satvaru, un to redzamību nosaka ne tikai modifikatori, bet arī konkrētā programmēšanas valoda un satvars.

Klases un objekta (instances) elements – **lauks**

Skats no «iekšpuses»

- **Objekts** sastāv no datu kopuma un šiem datiem piesaistītajām darbībām. Datus objektā reprezentē iekšējie mainīgie jeb ***lauki***.
- **lauks**
- **iekšējs mainīgais** *(member variable)*

EN field

LV lauks

ISO 2382 04.07.02, prot. Nr. 261 (13.01.2006)

LZA TK ITTEA protokoli

Klases un objekta (instances) elements – **metode**

Skats no «iekšpuses»

- **Objekts** sastāv no datu kopuma un šiem datiem piesaistītajām darbībām. Darbības reprezentē iekšējās funkcijas (un procedūras) jeb ***metodes***.
- **metode**
- **iekšēja funkcija** *(member function)*
- **funkcija** *(function)*

EN method
LV metode
prot. Nr. 567 (04.12.2020)
LZA TK ITTEA protokoli

Klases un objekta (instances) elements – **rekvizīts**

Skats no «iekšpuses»

- **Objekts** sastāv no datu kopuma un šiem datiem piesaistītajām darbībām.
  Datus objektā reprezentē iekšējie mainīgie jeb ***lauki***,
  bet darbības reprezentē iekšējās funkcijas (un procedūras) jeb ***metodes***.
- Taču, lai atvieglotu darbu ar objekta datiem
  daudzās programmēšanas valodās un satvaros tiek ieviests elements – **rekvizīts**.
- Rekvizīts ir īpaša veida elements, kas funkcionalitātē ir starpposms starp lauku un metodi.
  Piekļuve objekta rekvizītam izskatās tāpat kā piekļuve ieraksta laukam (strukturētajā programmēšanā),
  bet faktiski tiek īstenota, izmantojot metodes (funkcijas) izsaukumu.
  Mēģinot iestatīt rekvizīta vērtību, tiek izsaukta viena metode, bet, lai iegūtu rekvizīta vērtību, tiek izsaukta cita metode.
  Rekvizītam var iestatīt noklusējuma vērtību, kā arī norādīt, ka šis rekvizīts ir tikai lasāms.
  Parasti rekvizīts ir saistīts ar kādu objekta lauku, taču tas var nebūt saistīts ar nevienu objekta lauku,
  lai gan šī objekta lietotājs rīkojas ar rekvizītu tā, it kā tas būtu reāls lauks.

EN property

LV rekvizīts (objektorientētajā programmēšanā)

prot. Nr. 567 (04.12.2020)
LZA TK ITTEA protokoli

Klases un objekta (instances) elements – **atribūts**

Skats no «ārpuses»

- **attribute - atribūts**
- Klases un objekta (instances) elementu kontekstā bieži saskaramieas ar terminu **atribūts**.
- Terminu **atribūts** parasti lieto tajās situācijās, ja uz klases un objekta (instances) elementiem skatāmies no «ārpuses», t.i., ja nezinām, kurš ir lauks, metode un rekvizīts.

  EN attribute
  LV atribūts
  ISO 2382 17.02.12, prot. Nr. 300 (16.11.2007)
  LZA TK ITTEA protokoli

- Jāatzīmē, ka terminu ***lauks, iekšējais mainīgais, metode, rekvizīts*** un ***atribūts*** lietojums dažādos literatūras avotos un dažādu programmēšanas valodu un satvaru kontekstos nav konsekvents.

Klases mainīgais un instances (objekta) mainīgais (lauks)

- **instances mainīgais (instances lauks)**

  pieder tieši šim vienam konkrētam objektam (instancei)
  skat. analoģiju ar lokālajiem mainīgajiem

- **klases mainīgais (klases lauks)**

  ir viens uz visām konkrētās klases instancēm (objektiem) un eksistē pat tad, ja neviens objekts vēl neeksistē
  skat. analoģiju ar globālajiem mainīgajiem

Klases un instances mainīgo (lauku) salīdzinājums

Instances mainīgais

- Saistīts ar klases objektu
- Tiek deklarēts metodes __init()__ iekšpusē
- Piekļūst **`instance.mainīgais`**
- Atsevišķs katram konkrētam klases objektam

Klases mainīgais

- Saistīts ar klasi
- Tiek deklarēts klasē ārpus jebkuras metodes
- Piekļūst **`Klase.mainīgais`**
- Kopīgs visiem klases objektiem

Klases mainīgā un instances (objekta) mainīgā salīdzinājums

Klases mainīgais	***Instances mainīgais***
Klases manīgais ir viens uz visām klases instancēm (pat tad, ja vēl neviena klases instance nav izveidota). Daudzās programmēšanas valodās klases mainīgais tiek deklarēts, izmantojot modifikatoru ***static***.	Instances mainīgais pieder tieši šai instancei un ir unikāls katrai atsevišķai instancei
Klases mainīgais parasti tiek izveidots ikreiz, kad mēs sākam programmas izpildi.	Instances mainīgais parasti tiek izveidots ikreiz, kad mēs izveidojam atbilstošo klases instanci.
Klases mainīgais tam piešķirto vērtību parasti uzglabā (“atceras”) līdz programmas izpildes beigām.	Instances mainīgais tam piešķirto vērtību parasti uzglabā (“atceras”) tik ilgi, kamēr šī instance pastāv.
No “ārpuses” klases mainīgajam parasti piekļūst norādot klasi, kurai tas pieder, piemēram, *klase.mainīgais*	No “ārpuses” instances mainīgajam parasti piekļūst norādot instanci, kurai tas pieder, piemēram, *instance.mainīgais*

Un tagad paskatīsimies kā būtu «pareizāk» veidot klases, lai pašiem nebūtu pārsteigumi ar klases un instances mainīgajiem (laukiem)

```
class Auto:
    skaits = 0 # klases mainīgais – kopīgs visām klases instancēm un neizmantosim kā instances lokālo lauku

    def __init__(self, modelis="BMW X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis # instances lokālais lauks, kura vērtību padod caur argumentu
        self.gads = gads       # instances lokālais lauks, kura vērtību padod caur argumentu
        self.krasa = krasa     # instances lokālais lauks, kura vērtību padod caur argumentu
        self.atrums = 0   # instances lokālais lauks, kura vērtību padod kā konkrētu fiksētu vērtību

    def izdrukat(self):
        print("Automašīna:", self.marka, self.modelis, self.gads,
              self.krasa, self.atrums)

    def izmainit_klases_mainigo(self):
        Auto.skaits = Auto.skaits + 1

    def izmainit_instances_mainigo(self, paatrinajums):
        self.atrums = self.atrums + paatrinajums
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
0
0
0
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
Auto.skaits = 3
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
0
0
0
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
Auto.skaits = 3
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
0
0
0
```

```
3
3
3
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
Auto.skaits = 3
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
0
0
0

3
3
3
```

```
bembis1.skaits = 1
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
Auto.skaits = 3
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
0
0
0
3
3
3
```

```
bembis1.skaits = 1
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
1
3
3
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
Auto.skaits = 3
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
0
0
0
3
3
3
```

```
bembis1.skaits = 1
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
bembis2.skaits = 5
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
1
3
3
```

*# "interesanta" lauka **skaits** uzvedība*

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
```

```
bembis1 = Auto()
bembis2 = Auto()
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
Auto.skaits = 3
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
0
0
0
3
3
3
```

```
bembis1.skaits = 1
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
bembis2.skaits = 5
print(bembis1.skaits)
print(bembis2.skaits)
print(Auto.skaits)
```

```
1
3
3
1
5
3
```

Metodes

Klases mainīgais un instances (objekta) metode

- **instances metode**
  no tās iespējama piekļuve tieši šī objekta laukiem un citām instances metodēm
  (šādu metožu pirmais parametrs **`self`** ir norāde uz konkrētu klases instanci (objektu))

- Lauki un metodes var būt ne tikai piederošas konkrētam objektam (instancei),
  bet arī tādas, kas ir kopīgas visām šīs klases instancēm (objektiem) – klases elementiem

- **klases metode**
  no tās iespējama piekļuve tikai klases laukiem un citām klases metodēm, bet nevis konkrētā objekta elementiem (šādu metožu pirmais parametrs **`cls`** ir norāde uz klasi, nevis konkrētu klases instanci (objektu))

- **statiska metode**
  valodā *Python* statiskas metodes no klases metodēm atšķiras ar to,
  ka tās klasei pieder tikai klases strukturēšanas ietvaros,
  bet bez tiešas pieejas klases elementiem

Metožu salīdzinājums

Instances metode

- Nav dekoratora
- Pirmais parametrs **`self`**
- Izsauc **`instance.funkcija()`**
- Saistīta ar klases objektu
- Tā var mainīt objekta stāvokli
- Var piekļūt un modificēt gan klases, gan instanču mainīgos

Klases metode

- **`@classmethod`**
- Pirmais parametrs **`cls`**
- Izsauc **`Klase.funkcija()`**
- Saistīta ar klasi
- Tā var mainīt klases stāvokli
- Var piekļūt tikai klases mainīgajiem
- Izmanto fabricēšanas metožu veidošanai

Statiska metode

- **`@staticmethod`**
- Nav speciāla parametra
- Izsauc **`Klase.funkcija()`**
- Saistīta ar klasi
- Tā nevar mainīt klases vai objekta stāvokli
- Nevar piekļūt vai modificēt klases un instanču mainīgos

Un tagad paskatīsimies kā būtu «pareizāk» veidot klases, lai pašiem nebūtu pārsteigumi ar instances metodēm (funkcijām)

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa
        self.atrums = 0

    def izdrukat(self):
        print("Automašīna:", self.marka, self.modelis, self.gads,
              self.krasa, self.atrums)

    def izmainit_instances_mainigo(self, paatrinajums):
        self.atrums = self.atrums + paatrinajums

    def izmainit_klases_mainigo(self): # Var, bet labāk tā nedarīsim
        Auto.skaits = Auto.skaits + 1
```

Un tagad paskatīsimies kā būtu «pareizāk» veidot klases, lai pašiem nebūtu pārsteigumi ar klases metodēm (funkcijām)

```
class Auto:
    skaits = 0

    def __init__(self, modelis="X7", gads=2022, krasa="Bezkrāsas"):
        self.modelis = modelis
        self.gads = gads
        self.krasa = krasa

    @classmethod                    # maina klases mainīgo (lauku) vērtības
    def palielinat_skaitu(cls):
        cls.skaits = cls.skaits + 1

    @classmethod                    # darbojas kā veidojoša metode
    def izveidot_jaunu(cls, modelis, gads, krasa):
        return cls(modelis, gads + 1, "Sarkans")

bembis1 = Auto()
bembis2 = Auto.izveidot_jaunu("X3", 2020, "Melns")
print(bembis2.skaits, bembis2.modelis, bembis2.gads, bembis2.krasa)
Auto.palielinat_skaitu()
print(Auto.skaits)
print(bembis2.skaits, bembis2.modelis, bembis2.gads, bembis2.krasa)
```

Un tagad paskatīsimies kā būtu «pareizāk» veidot klases, lai pašiem nebūtu pārsteigumi ar statiskām metodēm

Mēs aplūkosim tikai gadījumus, kad veidojam klasi ar statiskām metodēm, lai veidotu pašu definēto funkciju tematiskos grupējumus

```
class Aritmetika:
    @staticmethod
    def saskaitit(a,b):
        return a+b

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

    @staticmethod
    def reizinat(a,b):
        return a*b

print(Aritmetika.saskaitit(7,5))
print(Aritmetika.atnemt(7,5))
print(Aritmetika.reizinat(7,5))
```