13. NODARBĪBA

PYTHON BRUŅRUPUČA GRAFIKA – MODULIS TURTLE

Python moduļi grafikas programmēšanai

Python ir vairāki moduļi, kas nodrošina grafikas un lietotāja grafiskās saskarnes (GUI) programmēšanas iespējas. Divi no šiem moduļiem ir iekļauti *Python* standarta bibliotēkā, un tie ir – **turtle** un **tkinter**.

Modulis **tkinter** ir paredzēts lietotāja grafiskās saskarnes veidošanai, piemēram, programmēšanas vide *Python IDLE* ir veidota, izmantojot **tkinter**.

Modulis **turtle** galvenokārt ir paredzēts grafisko objektu veidošanai, bet to var arī izmantot, lai veidotu vienkāršu grafisko saskarni.

Bruņrupuča grafika pirmo reizi tika izmantota 1960-tajos gados kā programmēšanas valodas *Logo* sastāvdaļa.

Kas ir bruņrupuča grafika?

Bruņrupuča grafikas ideja ir kustīga objekta – bruņrupuča - pārvietošanās pa laukumu. Objekta atrašanās vieta tiek aprakstīta ar koordinātām x, y.

Iedomāsimies, ka bruņrupucis pārvietojas, turot zīmuli. Zīmulis var būt gan pacelts uz augšu (nepieskaras laukumam), gan nolaists uz leju (pieskaras laukumam). Ja zīmulis pieskaras laukumam, tad, bruņrupucim pārvietojoties, tiek atstāta "aste" jeb zīmuļa nospieduma kontūra. Ja zīmulis ir pacelts, tad bruņurupuča pārvietošanās trajektorija nav redzama.

Bruņrupucim var dot dažādas komandas, nosakot tā atrašanās vietu, pārvietošanās virzienu un attālumu - tādejādi veidojot zīmējumu.

Modulis turtle

Lai uzsāktu darbu ar bruņrupuča grafiku, nepieciešams importēt moduli **turtle**.

```
import turtle
```

vai

```
from turtle import *
```

Funkcijas, kas iekļautas šajā modulī, var apskatīties *Python* čaulas logā ierakstot

```
>>> dir(turtle)
['Canvas', 'Pen', 'RawPen', 'RawTurtle', 'Screen', 'ScrolledCanvas', 'Shape', 'TK', 'TNavigator',
'TPen', 'Tbuffer', 'Terminator', 'Turtle', 'TurtleGraphicsError', 'TurtleScreen',
'TurtleScreenBase', 'Vec2D', '_CFG', '_LANGUAGE', '_Root',
...
'up', 'update', 'width', 'window_height', 'window_width', 'write', 'write_docstringdict', 'xcor',
'ycor']
```

Lai aplūkotu, kādas funkcijas aprakstu tuvāk, izmanto funkciju *help*.

```
>>> help(turtle.left)
Help on function left in module turtle:

left(angle)
   Turn turtle left by angle units.

   Aliases: left | lt

   Argument:
   angle -- a number (integer or float)

   Turn turtle left by angle units. (Units are by default degrees,
   but can be set via the degrees() and radians() functions.)
   Angle orientation depends on mode. (See this.)

   Example:
   >>> heading()
   22.0
   >>> left(45)
   >>> heading()
   67.0
```

Funkcijas, kas atrodas modulī **turtle**, var aplūkot vietnē:

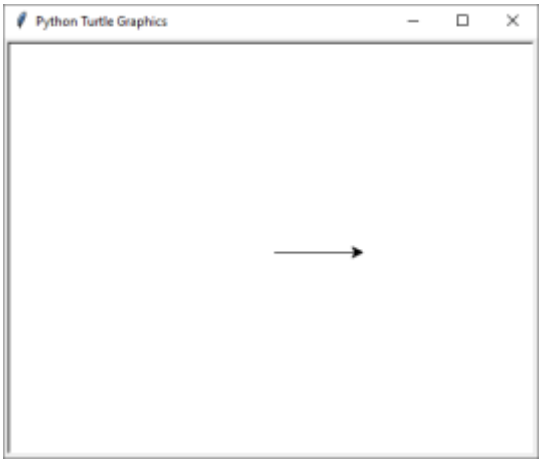
https://docs.python.org/3.3/library/turtle.html?highlight=turtle#module-turtle

Pirmā turtle programma

Pēc moduļa **turtle** importēšanas var sāk veidot bruņrupuča grafikas programmu.

```
import turtle

turtle.forward(80)
```



Programma grafiskajā logā uzzīmē 80 pikseļu līniju.

Kā var redzēt bruņrupucis sāka savu kustību no loga centra (noklusējuma koordinātas (0;0)) un pārvietojās 80 soļus jeb pikseļus pa labi.

Pēc noklusējuma bruņrupuča forma ir "bultiņa" un tā virziens ir – pa labi.

Nedaudz vēlāk apskatīsimies, kā mainīt bruņrupuča īpašības.

Grafiskā loga izveidošana

Pirmajā piemērā Jūs redzējāt, ka, uzsākot bruņrupuča pārvietošanu (zīmēšanu), grafiskais logs tika izveidots automātiski.

Grafiskais logs ir objekts, un, kā zināms, programmēšanā objektam piemīt gan īpašības, gan darbības (metodes). Lai varētu mainīt loga īpašības, nepieciešams iegūt atsauci uz šo logu (objektu).

Atsauci uz grafisko logu iegūst ar metodi **Screen**.

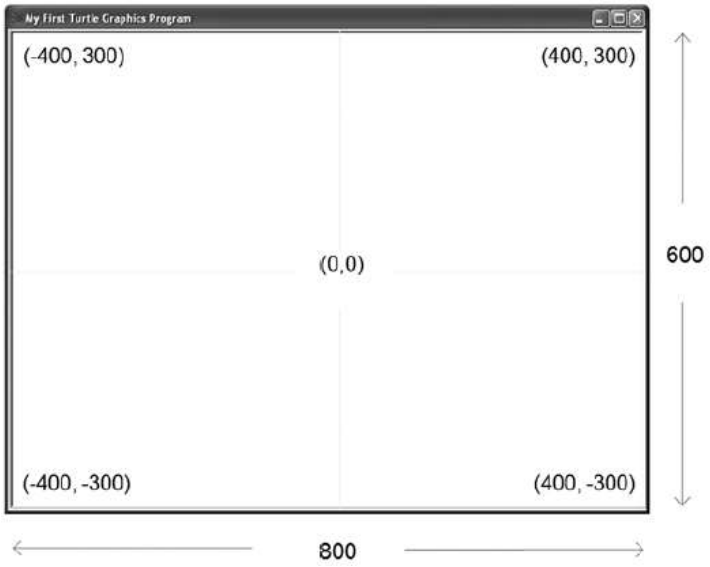
Piemēram,

```
import turtle

# izveido grafikas logu, ieguust atsauci uz so logu
window = turtle.Screen()
# norada loga izmeru
window.setup( 800, 600 )
# norada loga virsrakstu
window.title( 'My First Turtle Graphics Program' )
```

Šajā piemērā tiek izveidots grafiskais logs un mainīgajā **window** ierakstīta atsauce uz šo logu (objektu). Pēc tam tiek norādīts loga izmērs (metode **setup**) un virsraksts (metode **title**).

Izveidotam logam koordinātu sākumpunkts (0;0) atrodas loga centrā.



Loga fona krāsa pēc noklusējuma ir balta. Loga krāsu var izmainīt ar metodi **bgcolor**.

```
window.bgcolor('blue')
```

Beidzot programmu, grafikas logu var aizvērt ar metodi **exitonclick** vai **bye**. Metode **bye** logu aizver automātiski.

```
turtle.exitonclick()
```

Visas moduļa **turtle** loga (screen) metodes var aplūkot šeit:

https://docs.python.org/3/library/turtle.html#methods-of-turtlescreen-screen

Bruņrupucis

Tāpat kā grafiskais logs, arī bruņrupucis ir *Python* objekts.

Izveidojot grafisko logu, tiek izveidots "noklusētais" bruņrupucis. Sākuma novietojums noklusētajam bruņrupucim ir loga centrā (0,0) un tā forma ir "bultiņa".

Atsauci uz noklusēto bruņrupuci var iegūt, izmantojot metodi **getturtle**.

```
felix = turtle.getturtle()  # iegust atsauci uz brunrupuci
```

Bruņrupuča sākuma atrašanās vietu var noteikt ar metodi **setup**.

```
turtle.setup( width, height, x, y )
```

Programmas kods:

```
import turtle

window = turtle.Screen()
window.setup( 600, 400 )
window.title('My First Turtle Graphics Program' )

felix = turtle.getturtle()

turtle.exitonclick()
```

Rezultāts

Bruņrupuča īpašības un uzvedība

Kā jau tika minēts, jebkuram objektam piemīt gan īpašības, gan uzvedība.

Bruņrupuča objektam ir trīs galvenās īpašības jeb atribūti: **novietojums** (*possition*), **virziens** (*heading*) un **pildspalva** (*pen*).

Novietojums

Tikko izveidotam bruņrupucim novietojums ir (0, 0), t.i., loga centrā. Novietojumu var mainīt, norādot absolūto pozīciju vai relatīvo pozīciju.

Bruņrupuča atrašanās vietu iegūst ar metodi **position**, tā atgriež objekta atrašanās vietas koordinātes kā kortežu (*tuple*).

Absolūtās atrašanās vietas norādīšana

Absolūto atrašanās vietu nosaka ar metodi **setposition**.

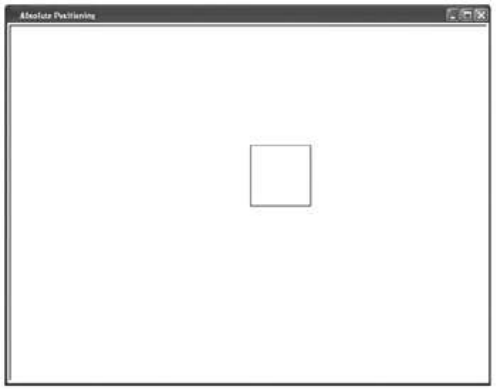
Jāatceras, ka, pārvietojot bruņrupuci, tiks zīmēta tā trajektorija, ja vien pildspalva nebūs pacelta (**penup**).

Kods

```
# iegust atsauci uz brunrupuci
felix = turtle.getturtle()
felix.hideturtle()

# izveido kvadratu
felix.setposition( 100, 0 )
felix.setposition( 100, 100 )
felix.setposition( 0, 100 )
felix.setposition( 0, 0 )
```

Rezultāts



Relatīvās pārvietojuma norādīšana

Bruņrupuča atrašanās vietu var mainīt, norādot tā relatīvo pārvietojumu.

Relatīvo pārvietojumu norāda, izmantojot **pārvietojuma attālumu** un **virzienu** – bruņrupucis pārvietosies noteiktu soļu skaitu norādītajā virzienā.

Pārvietojuma attālumu norādītajā virzienā nosaka metode **forward**.

Tikko izveidotam bruņrupucim virziens ir 0 grādi (pa labi).

Virziens 90 grādi nosaka pārvietojumu uz augšu, 180 grādi nosaka pārvietojumu pa kreisi, virziens 270 grādi nosaka pārvietojumu uz leju.

Pašreizējo bruņrupuča virzienu var noteikt ar metodi **heading**.

Virzienu var noteikt, izmantojot metodi **setheading**.

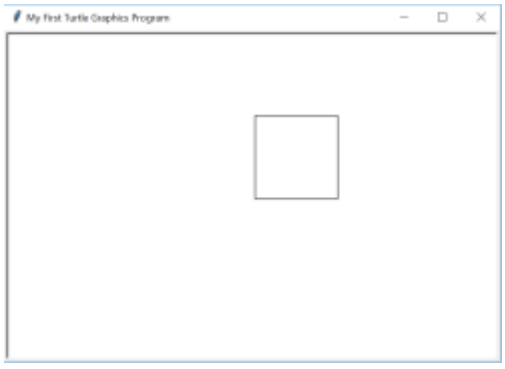
Virzienu var mainīt, izmantojot metodes **right** un **left**, kas maina virzienu par noteiktu grādu skaitu attiecībā pret iepriekšējo virzienu (relatīva virziena maiņa).

Programmas kods

```
felix = turtle.getturtle()
felix.hideturtle()

# izveido kvadratu, mainot relativo poziciju
felix.forward( 100 )
felix.left( 90 )
felix.forward( 100 )
felix.left( 90 )
felix.forward( 100 )
felix.left( 90 )
felix.forward( 100 )
```

Rezultāts



```
felix.left( 90 )
```

Pārvietošanai pretējā virzienā var izmantot arī metodi **backward**.

Vairāk par funkcijām, kas nosaka bruņrupuča pārvietošanos:

https://docs.python.org/3.3/library/turtle.html?highlight=turtle#turtle.forward

Pildspalva

Bruņrupuča atribūts "pildspalva" **pen** nosaka zīmēšanas iespējas.

Vairāk par pildspalvas funkcijām šeit:

https://docs.python.org/3.3/library/turtle.html?highlight=turtle#turtle.pendown

Pildspalva var būt pacelta uz augšu (metode **penup**) vai nolaista uz leju (**pendown**). Ja pildspalva ir pacelta, tad bruņrupuča pārvietojuma kontūra nav redzama.

Pildspalvas kontūras biezumu nosaka metode **pensize**. Biezums tiek noteikts pikseļos.

Pildspalvas krāsa

Pēc noklusējuma pildspalvas krāsa ir melna. Krāsu var mainīt, izmantojot funkciju **pencolor**. Funkcijas arguments ir krāsa, ko var uzdot dažādi:

- Virkne ar krāsas nosaukumu, piemēram "red".
- Virkne, kas satur RGB kodu heksadecimālā pierakstā, piemēram "#FF0000" (sarkana). Virkne sākas ar simbolu #, tālāk seko seši heksadecimālie cipari, no kuriem pirmie divi atbilst sarkanās krāsas komponentei, nākamie divi atbilst zaļās krāsas komponentei un pēdējie divi atbilst zilās krāsa komponentei.
- RGB komponenšu vērtības, kas var būt no 0 līdz 255, piemēram, 255, 0, 0

Dažas biežāk izmantotās krāsas ir: 'white', 'red', 'blue', 'green', 'yellow', 'grey', 'black'.

Pēc noklusējuma krāsas jāuzdod ar virkni.

```
felix.pencolor('red')  # sarkana
```

Izmantojot funkciju **colormode**, var noteikt krāsas uzdošanas veidu: ar virkni vai ar 3 RGB komponentēm. Piemēram,

```
turtle.colormode( 255 )
felix.pencolor( 238, 130, 238 )  # violeta
```

Funkcijas **colormode** sintakse ir:

```
turtle.colormode(cmode=None)
```

Arguments cmode var būt 1.0 vai 255. 255 nozīmē krāsu uzdošanu ar 3 RGB komponentēm.

Saraksts ar iespējamiem krāsu nosaukumiem un tām atbilstošām RGB vērtībā ir pieejams šeit: http://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm

Citi tīmeklī pieejamie krāsu rīki:

http://cloford.com/resources/colours/500col.htm

http://www.colr.org/

http://paletton.com/#uid=1000u0kllllaFw0g0qFqFg0w0aF

Citas bruņrupuča īpašības

Papildus aplūkotajām īpašībām, ir vēl citas īpašības, kas palīdz kontrolēt bruņrupuča uzvedību. Tās ir īpašības, kas nosaka bruņrupuča redzamību, izmēru, formu, aizpildījuma krāsu, ātrumu.

Bruņrupuča redzamība

Bruņrupuča redzamību nosaka metodes **showturtle** un **hideturtle**.

Metode **isvisible** atgriež bruņrupuča redzamības stāvokli.

Metodes, kas nosaka bruņrupuča redzamību pieejamas šeit:

https://docs.python.org/3/library/turtle.html#turtle.showturtle

Bruņrupuča izmērs

Bruņrupuča formas izmēru kontrolē divas metodes **resizemode** un **shapesize** (**turtlesize)**.

Metode **resizemode** nosaka veidu, kā varēs mainīt izmēru. Metodei **resizemode** iespējamas trīs dažādas argumenta vērtības:

resizemode('user')	Ļauj lietotājam (programmētājam) noteikt bruņrupuča izmēru, izmantojot metodi **turtlesize**. Pretējā gadījumā metodes **turtlesize** izmantošanai nebūs nekādas nozīmes.
resizemode('auto')	Bruņrupuča izmēru noteiks automātiski atbilstoši pildspalvas izmēram.
resizemode('noresize')	Bruņrupuča izmērs paliks nemainīgs.

Metode **shapesize** (**turtlesize)** nosaka bruņrupuča izmēru.

shapesize(x, y) turtlesize(x, y)	x – nosaka bruņrupuča izmēru horizontālā virzienā; y - nosaka bruņrupuča izmēru vertikālā virzienā. x un y ir palielinājuma koeficienti.
shapesize(x, y, b) turtlesize(x, y, b)	b nosaka bruņrupuča formas ārējās kontūras biezumu.

Piemēram

```
>>> turtle.shapesize()
(1.0, 1.0, 1)
>>> turtle.resizemode("user")
>>> turtle.shapesize(5, 5, 12)
>>> turtle.shapesize()
(5, 5, 12)
>>> turtle.shapesize(outline=8)
>>> turtle.shapesize()
(5, 5, 8)
```

Par metodēm **resizemode** un **shapesize** vairāk šeit:

https://docs.python.org/3/library/turtle.html#turtle.resizemode

https://docs.python.org/3/library/turtle.html#turtle.shapesize

Bruņrupuča forma

Ir vairāki veidi, kā noteikt bruņrupuča formu un arī aizpildījuma krāsu.

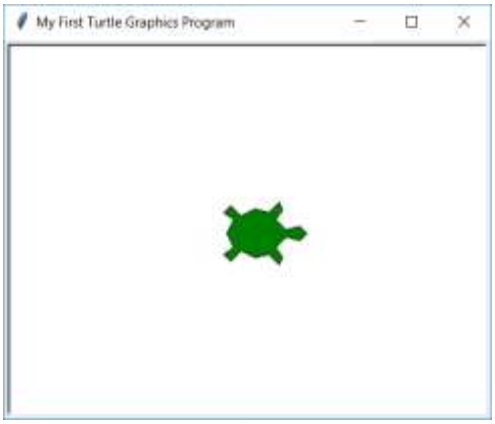
Bruņrupuča formu nosaka ar metodi **shape**.

Metodei **shape** kā parametru iedod virkni ar kādu no iepriekšdefinētiem formu veidiem: ‘arrow’, ‘turtle’, ‘circle’, ‘square’, ‘triangle’ un ‘classic’ (noklusētā bruņrupuča forma).

Bruņrupuča aizpildījuma krāsu nosaka ar metodi **fillcolor**.

Savukārt metode **color** ar diviem argumentiem vienlaikus nosaka gan kontūras, gan aizpildījuma krāsu.

```
felix = turtle.getturtle()
felix.shape('turtle')
felix.fillcolor('green')
felix.resizemode('user')
felix.turtlesiz(3, 3)
```



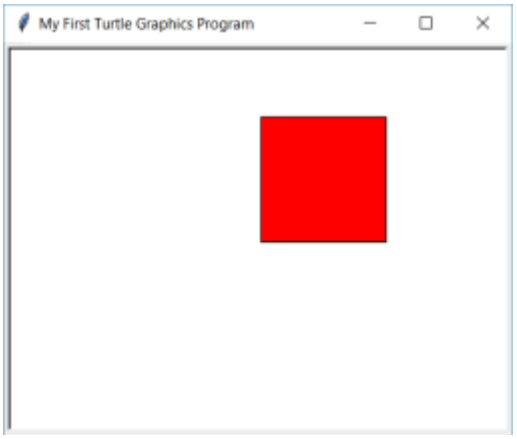
Vairāk par bruņrupuča krāsām.

https://docs.python.org/3/library/turtle.html#turtle.color

Zīmējuma aizpildīšana ar krāsu

Lai aizpildītu ar krāsu uzzīmētu zīmējumu, pirms zīmēšanas, izsauc metodi **begin_fill**, pēc tam uzzīmē zīmējumu un izsauc metodi **end_fill**.

```
# izveido aizpildiitu kvadratu
felix.color( 'black', 'red' ) # fona kraasa, aizpildijuma krasa
felix.begin_fill()
felix.setposition( 100, 0 )
felix.setposition( 100, 100 )
felix.setposition( 0, 100 )
felix.setposition( 0, 0 )
felix.end_fill()
```



Metode **filling** atgriež zīmējuma aizpildīšanas statusu (**True** – ja ir iestatīta aizpildīšana, **False** – pretējā gadījumā).

Vairāk par metodēm, kas nosaka zīmējuma aizpildīšanu:

https://docs.python.org/3/library/turtle.html#turtle.filling

Bruņrupuča formas

Iespējams veidot jaunas bruņrupuča formas un pievienot tās formu vārdnīcai (*shape dictionary*).

Formu var izveidot ar *Python* iespējām vai izmantot jau gatavus attēlus .gif.

https://docs.python.org/3/library/turtle.html#turtle.getshapes

Formas izveidošana

Apskatīsim formas izveidošanas piemēru, izveidojot daudzstūri, norādot tā virsotņu koordinātas.

```
# iegust atsauci uz brunrupuci
felix = turtle.getturtle()

# izveido formu ar nosaukumu 'mypologon'
# ieraksta to formu bibliotekaa
turtle.register_shape( 'mypoligon', ((0,0), (100, 0), (140, 40)) )
# pieskir brunrupucim izveidoto formu
felix.shape( 'mypoligon' )
felix.fillcolor( 'white' )
```

Izveidotās formas pievienošana

Metode **register_shape** ieraksta formu bibliotēkā jaunu formu (**new_shape**), piešķirot tai nosaukumu (‘**shape_name’**).

Metodes **register_shape** vispārīgā forma:

```
register_shape( ‘shape_name’, new_shape )
```

Iepriekš aplūkotajā piemērā jaunā forma tiek uzdota, kā daudzstūra virsotņu koordināšu kortežs (*tuple*).

Kad forma ar norādīto vārdu ir pievienota bibliotēkai, to var izmantot, lai piešķirtu bruņrupucim šo formu.

Ir iespējams veidot arī sarežģītākas formas, kas sastāv no vairākiem daudzstūriem (*compound shapes*).

Iepriekš izveidotā forma varētu šķist ne pārāk pievilcīga, tomēr ar tās palīdzību var veidot interesantus attēlus.

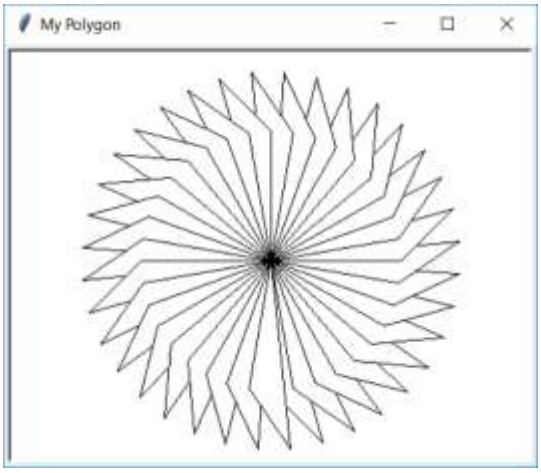
Piemēram,

```
# iegust atsauci uz brunrupuci
felix = turtle.getturtle()

# izveido formu ar nosaukumu 'mypologon'
# ieraksta to formu bibliotekaa
turtle.register_shape( 'mypoligon', ((0,0), (100, 0), (140, 40)) )

# pieskir brunrupucim izveidoto formu
felix.shape( 'mypoligon' )
felix.fillcolor( 'white' )

for angle in range( 0, 360, 10 ):
   felix.setheading( angle )
   felix.stamp()
```



Šajā piemērā metode **stamp** atstāj formas nospiedumu. Katrā cikla iterācijā forma tiek pagriezta, tādejādi iegūstot zīmējumu.

Bruņrupuča formas izveidošana, izmantojot attēlu

Cits veids, kā izveidot bruņrupuča formu, ir izmantot attēlu. Attēla failam jābūt ar paplašinājumu .gif. Pēc tam šo attēlu reģistrē un ar metodes **shape** palīdzību piešķir formu bruņrupucim.

Šādi izveidotu formu nevar rotēt.

```
register_shape( ‘image1.gif’)
felix.shape( ‘image1.gif’ )
```

Bruņrupuča ātrums

Bruņrupuča pārvietošanās ātrumu nosaka ar funkcijas **speed** palīdzību. Ātruma iespējamās vērtības ir no 0 līdz 10, kur 6 ir "normāls" ātrums.

Tā vietā, lai ātrumu norādītu ar skaitļiem, var izmantot aprakstošas vērtības:

10: 'fast', 6: 'normal', 3: 'slow', 1: 'slowest', 0: 'fastest'

Ja bruņrupuci izmanto tikai zīmēšanai, tad izvēlas lielu ātrumu.

Zīmējuma izdzēšana

Lai izdzēstu uzzīmēto, izmanto metodi **reset**. Metode **reset** izdzēš uzzīmēto un novieto bruņrupuci sākuma pozīcijā.

```
felix.reset()
```

Zīmējuma dzēšanai var izmantot arī metodi **clear**, tā izdzēš uzzīmēto un atstāj bruņrupuci tā pašreizējā vietā.

```
felix.clear()
```

Vairāku bruņrupuču izveidošana

Līdz šim mēs izmantojām tikai vienu – noklusēto bruņrupuci. Tomēr ir iespējams izveidot pēc patikas daudz bruņrupuču. Jaunu bruņrupuci izveido ar metodi **Turtle**.

```
turtle1 = turtle.Turtle()
turtle2 = turtle.Turtle()
```

Bruņrupuču uzglabāšanai var izmantot sarakstu.

```
turtles = []
turtles.append(turtle.Turtle())
turtles.append(turtle.Turtle())
turtles.append(turtle.Turtle())
```