Python Turtle parancslista a Kovács Mihály Országos Grafikus Programozási versenyhez[1]

Tartalom

[1] Forrás: https://docs.python.org/3/library/turtle.html

Python turtle beállítása

Default turtle mód

Minden turtle utasítást bemásol a névtérbe, előnye, hogy rövidebbek az utasítások (nem kell a `turtle.` előtag a teknőc utasítások elé. Hosszabb programok és animációk készítésekor nem előnyös.

Példa

```
from turtle import *

forward(100)
left(90)
forward(100)
```

Normál turtle mód

A turtle modul egyben kerül importálásra, mindent a `turtle.` előtaggal érsz el, ami hosszabb kódot eredményez, de kevesebb hibát okoz. A továbbiakban ezt a módot tárgyaljuk.

Példa

```
import turtle

t = turtle.Turtle()
t.forward(100)

screen = turtle.Screen()
screen.bgcolor("white")
```

Háttért tulajdonságainak beállítása

colormode(mód)

A `colormode()` a turtle-ben azt határozza meg, hogyan értelmezze a színeket numerikusan (RGB értékekkel). Azt mondja meg a turtle-nek, hogy az RGB értékek:

- `mód = 1.0`**: 0–1 között,**
- `mód = 255`: **0–255 között** legyenek.

Példa

```
import turtle
```

```
screen = turtle.Screen()
t = turtle.Turtle()

screen.colormode(1.0)
screen.bgcolor(0.5, 0.2, 0.8)
t.color(0.0, 0.0, 1.0)

screen.colormode(255)
screen.bgcolor(128, 52, 202)
t.color(0, 0, 255)
```

bgcolor() | bgcolor(szín) | bgcolor(r, g, b)

A TurtleScreen háttérszínének lekérdezése vagy beállítása.

bgcolor()

Visszaadja az aktuális háttérszínt

- szöveges színleírásként **(pl. "orange")**
- vagy **RGB tuple-ként** (r, g, b)

A visszakapott érték felhasználható más `color()`, `pencolor()`, `fillcolor()` vagy `bgcolor()` hívásoknál.

bgcolor(szín)

Beállítja a háttérszínt a megadott színre. A szín egy szöveg, amely lehet:

- Előre definiált szín: pl. `"red", "yellow"`
- A szín hexadecimális kódja: `"#33cc8c"`

bgcolor(r, g, b)

Ugyanaz, mint az előző, csak az RGB értékeket külön paraméterként adod meg. Az RGB megadási módja függ a `colormode(mód)` beállítástól.

Példa

```
screen.bgcolor("orange")
screen.bgcolor()
# 'orange'
screen.bgcolor("")
screen.bgcolor()

screen.bgcolor("#800080")
screen.bgcolor()
```

```
# (128, 0, 128) / (0.5019607843137255, 0.0, 0.5019607843137255)
colormode(255)
screen.bgcolor(128, 0, 128)
colormode(1.0)
screen.bgcolor(0.5, 0.0, 0.5)
```

bgpic() | bgpic(képnév)

Paraméter nélkül az aktuális háttérkép nevének lekérdezése. A képnév paraméterben megadott GIF kiterjesztésű kép beállítása háttérképnek.

Példa

```
import turtle
screen = turtle.Screen()
screen.bgpic() # háttérkép lekérdezése
# "nopic"
screen.bgpic("hatter.gif") # a hatter.gif beállítása hátérképnek
screen.bgpic("nopic") # háttérkép eltávolítása
```

clearscreen()

Mindent töröl a képernyőről: az összes rajzot, az összes teknőcöt, az összes eseménykezelést. Visszaállítja az alapállapotot: fehér háttér, nincs háttérkép, nincs billentyű/kattintás esemény, a rajzolás ismét be van kapcsolva.

Példa

```
import turtle
screen = turtle.Screen()
screen.clearscreen()
```

resetscreen()

Eltávolít minden korábbi rajzot, törli az összes teknőcöt és megszünteti az összes eseménykezelést (billentyű- és egérkattintásokat). A képernyő visszaáll az alapértelmezett állapotba: fehér háttérrel, háttérkép nélkül, és a rajzolás ismét aktív.

Röviden: olyan, mintha a program most indult volna el.

Minden teknőcöt alapállapotba állít a képernyőn.

screensize() | screensize(szélesség, magasság, szín)

Paraméter nélkül visszaadja az aktuális vászonméretet, paraméterekkel átméretezi **a rajzolási vásznat**, **nem** az ablakot!

Paraméterek:

- `szélesség` – vászon szélessége pixelben
- `magasság` – vászon magassága pixelben
- `szín` – háttérszín (szöveg vagy RGB tuple – lásd `. bgcolor() | bgcolor(szín)`)

Példa

```
screen.screensize()
# (400, 300)
screen.screensize(2000, 1500)
screen.screensize()
# (2000, 1500)
```

Teknőc mozgatás

forward(távolság) | fd(távolság)

A teknőc előre mozog `távolság` pixellel.

```
import turtle

t = turtle.Turtle()
t.forward(100)
```

backward(távolság) | back(távolság) | bk(távolság)

A teknőc hátra mozog a távolság paraméterben megadott pixelek számával.

```
import turtle

t = turtle.Turtle()
t.backward(100)
```

right(szög) | rt(szög)

A right(szög) vagy röviden rt(szög) utasítás a teknőst jobbra fordítja a megadott szöggel.

- A szög mértékegysége fok
- A teknős helyzete nem változik, csak az iránya

Példa

```
turtle.right(90)
```

left(szög) | lt(szög)

A left(szög) vagy lt(szög) utasítás a teknőst balra fordítja a megadott szöggel.

- A szög értéke fokban van megadva

Példa

```
turtle.left(45)
```

goto(x, y) | setpos(x, y) | setposition(x, y)

A goto(x, y), setpos(x, y) és setposition(x, y) utasítások a teknőst egy megadott koordinátára helyezik át.

- Az x és y a képernyő koordinátái,
- Ha a toll lent van, a teknős vonalat rajzol mozgás közben.

Példa

```
turtle.goto(100, 50)
```

home()

A home() utasítás a teknőst visszaviszi a kezdőpontra.

- Kezdőpont: (0, 0)
- Alapértelmezett irány: kelet (0 fok)

Példa

```
turtle.home()
```

setheading(irány) | seth(irány)

A setheading(irány) vagy seth(irány) utasítás a teknős irányát állítja be.

Az irány fokban van megadva

- 0 fok: jobbra (kelet)
- 90 fok: felfelé (észak)
- 180 fok: balra (nyugat)
- 270 fok: lefelé (dél)

Példa

```
turtle.setheading(90)
```

setx(x)

A setx(x) utasítás a teknős x koordinátáját változtatja meg, miközben az y koordináta változatlan marad.

Példa

```
turtle.setx(150)
```

sety(y)

A sety(y) utasítás a teknős y koordinátáját változtatja meg, miközben az x koordináta változatlan marad.

Példa

```
turtle.sety(-100)
```

Teknőd állapotainak lekérdezése

position() | pos()

A **position()** vagy röviden **pos()** függvény visszaadja a teknős **aktuális pozícióját**.

- Az eredmény egy (x, y) koordinátapár (tuple)

Példa

```
hely = turtle.pos()
print(hely)
```

towards()

A **towards()** függvény meghatározza azt az **irányt (fokban)**, amerre a teknősnek fordulnia kellene egy adott ponthoz vagy egy másik teknőshöz képest.

- A visszatérési érték fokban megadott irány
- A teknős irányát nem változtatja meg

Példa

```
irany = turtle.towards(100, 0)
print(irany)
```

xcor()

A **xcor()** függvény visszaadja a teknős **aktuális x koordinátáját**.

Példa

```
x = turtle.xcor()
```

```
print(x)
```

ycor()

A **ycor()** függvény visszaadja a teknős **aktuális y koordinátáját**.

Példa

```
y = turtle.ycor()
print(y)
```

heading()

A **heading()** függvény visszaadja a teknős **aktuális irányát** fokban.

- 0°: jobbra (kelet)
- 90°: felfelé (észak)
- 180°: balra (nyugat)
- 270°: lefelé (dél)

Példa

```
irany = turtle.heading()
print(irany)
```

---

distance(x, y) | distance(teknős)

A `distance()` függvény kiszámítja a teknős távolságát egy megadott ponttól vagy egy másik teknőstől. Az eredmény lebegőpontos szám

Példa

```
tav = turtle.distance(0, 0)
print(tav)
```

Rajzolás tulajdonságainak beállítása

speed() | speed(n)

A `speed()` utasítás a teknőc rajzolási sebességét kérdezi le vagy állítja be. Értéke 0 és 10 között lehet, ahol a nagyobb szám gyorsabb mozgást jelent. A 0 a lehető leggyorsabb rajzolást állítja be (animáció nélkül).

Példa

```
import turtle
t = turtle.Turtle()

t.speed(1)   # nagyon lassú
t.forward(100)
```

tracer()

A `tracer()` utasítás az animáció frissítését szabályozza. Segítségével gyorsíthatjuk a rajzolást azzal, hogy nem minden lépésnél frissül a képernyő. Lásd még: `tracer() | tracer(n) | tracer(n, idő)`

Példa

```
import turtle
turtle.tracer(0)   # animáció kikapcsolása

t = turtle.Turtle()
for i in range(100):
    t.forward(5)
    t.right(5)

turtle.update()    # képernyő frissítése
```

reset()

A `reset()` utasítás törli az adott teknőc rajzait, és visszaállítja alaphelyzetbe. A teknőc visszakerül a kezdőpontra, alap irányba és alap beállításokkal.

Példa

```
import turtle
t = turtle.Turtle()

t.forward(100)
t.reset()   # rajz törlése, teknőc alaphelyzetbe
```

clear()

A `clear()` utasítás csak a rajzot törli, de a teknőc helyzetét nem változtatja meg. A teknőc ott marad, ahol volt, csak a vonalak tűnnek el.

Példa

```
import turtle
t = turtle.Turtle()

t.forward(100)
t.clear()   # rajz eltűnik, teknőc marad
```

write(szöveg) | write(szöveg, mozgat)

A `write()` utasítás szöveget ír ki a képernyőre a teknőc aktuális helyén. Használható szöveg megjelenítésére, feliratok készítésére. Ha `mozgat` igaz, a toll a szöveg jobb alsó sarkára kerül. Megadható betűtípus is a `font` kulcsszavas paraméterrel

Példa

```
import turtle
t = turtle.Turtle()

t.write("Hello Turtle!", font=("Arial", 16, "normal"))
```

Toll tulajdonságok

pendown() | pd() | down()

A `pendown()` utasítás bekapcsolja a rajzolást. Amikor a teknőc mozog, vonalat hagy maga után.

Példa

```
import turtle
t = turtle.Turtle()

t.pendown()
t.forward(100)
```

penup() | pu() | up()

A `penup()` utasítás kikapcsolja a rajzolást. A teknőc mozog, de nem rajzol vonalat.

Példa

```
import turtle
t = turtle.Turtle()

t.penup()
```

```
t.forward(100)
```

`pensize()` | `width()` | `pensize(n)` | `width(n)`

A `pensize()` utasítás a vonal vastagságát állítja be. Minél nagyobb az érték, annál vastagabb a vonal.

Példa

```
import turtle
t = turtle.Turtle()

t.pensize(5)
t.forward(100)
```

`pen(**beállítások)`

A `pen()` utasítás a toll aktuális állapotát kezeli. Segítségével egyszerre több tollbeállítást is megadhatunk vagy lekérdezhetünk. A beállítások nevei megegyeznek a hozzájuk tartozó saját eljárások neveivel.

Példa

```
import turtle
t = turtle.Turtle()

t.pen(pencolor="red", pensize=3)
t.forward(100)
```

`isdown()`

Az `isdown()` megmondja, hogy a toll éppen rajzol-e. Igaz (`True`), ha a toll lent van, hamis (`False`), ha fent.

Példa

```
import turtle
t = turtle.Turtle()

print(t.isdown())    # True
t.penup()
print(t.isdown())    # False
```

`color()` | `color(vonalszín)` | `color(vonalszín, kitöltszín)`

A `color()` utasítás egyszerre állítja a vonal- és kitöltési színt. Egy vagy két színt is megadhatunk.

Példa

```
import turtle
t = turtle.Turtle()

t.color("blue", "yellow")
t.begin_fill()
t.circle(50)
t.end_fill()
```

`pencolor()` | `pencolor(szín)`

A `pencolor()` a rajzolóvonal színét állítja be.

Példa

```
import turtle
t = turtle.Turtle()

t.pencolor("green")
t.forward(100)
```

`fillcolor()` | `fillcolor(szín)`

A `fillcolor()` az alakzat kitöltési színét állítja be.

Példa

```
import turtle
t = turtle.Turtle()

t.fillcolor("red")
t.begin_fill()
t.circle(50)
t.end_fill()
```

`filling()`

A `filling()` megmondja, hogy éppen folyamatban van-e a kitöltés. Igaz (`True`), ha a `begin_fill()` már megtörtént, de az `end_fill()` még nem.

Példa

```
import turtle
t = turtle.Turtle()

t.begin_fill()
print(t.filling())    # True
t.end_fill()
print(t.filling())    # False
```

begin_fill()

A `begin_fill()` jelzi a kitöltés kezdetét. Az ezután rajzolt alakzat kitöltésre kerül.

Példa

```
import turtle
t = turtle.Turtle()

t.begin_fill()
t.circle(50)
```

end_fill()

Az `end_fill()` lezárja a kitöltést és kiszínezi az alakzatot.

Példa

```
import turtle
t = turtle.Turtle()

t.fillcolor("purple")
t.begin_fill()
t.circle(50)
t.end_fill()
```

Kész rajzoló eljárások

circle(r, a)

A `circle()` utasítás kört vagy körívet rajzol. Paraméterként megadható a kör sugara (`r`), illetve opcionálisan a rajzolt ív szöge (`a`).

Példa

```
import turtle
t = turtle.Turtle()

t.circle(50)      # teljes 50 sugarú kör
t.circle(50, 180) # 50 sugarú félkör
```

dot() | dot(méret) | dot(méret, szín)

A `dot()` utasítás egy kitöltött pontot rajzol a teknőc aktuális helyén. Megadható a pont mérete és színe is.

Példa

```
import turtle
t = turtle.Turtle()

t.dot(20, "red")
```

stamp()

A `stamp()` utasítás lenyomatot készít a teknőc aktuális alakjáról. A lenyomat a helyén marad akkor is, ha a teknőc elmozdul vagy eltűnik. A függvény visszatérési értéke a lenyomat azonosítója.

Példa

```
import turtle
t = turtle.Turtle()

stamp_id = t.stamp()
t.forward(100)
```

clearstamp(azon)

A `clearstamp()` egy korábban létrehozott lenyomatot töröl. A törléshez meg kell adni a lenyomat azonosítóját.

Példa

```
import turtle
t = turtle.Turtle()

stamp_id = t.stamp()
t.forward(100)
```

```
t.clearstamp(stamp_id)
```

clearstamps() | clearstamps(n)

A `clearstamps()` az összes (vagy megadott számú) lenyomatot törli. Paraméter nélkül az összes lenyomat eltűnik. Ha az `n` paraméter pozitív, az első `n`, ha negatív, az utolsó `n` lenyomatot törli.

Példa

```
import turtle
t = turtle.Turtle()

for i in range(5):
    t.stamp()
    t.forward(30)

t.clearstamps()
```

Animáció

no_animation()

Ideiglenesen kikapcsolja a turtle animációt. A `no_animation` blokkban lévő kód **nem animálva** fut le, és **csak a blokk végén jelenik meg egyszerre a rajz**. (**Python 3.14-től érhető el.)**

Ez nagyon hasznos, ha:

- gyorsítani akarod a rajzolást
- nem akarod látni a „rajzolás közbeni” mozgást

Példa

```
with screen.no_animation():
    for dist in range(2, 400, 2):
        fd(dist)
        rt(90)
```

A spirál **egyből kirajzolódik**, nem lépésenként.

delay() | delay(idő)

Beállítja vagy lekérdezi a rajzolás késleltetését. Minél nagyobb a késleltetés értéke, annál lassabb az animáció, az `idő` paraméter pozitív egész szám, a két képernyőfrissítés közti idő ms-ban.

Példa

```
screen.delay()
# 10 - jelenlegi késleltetési idő: 10 ms
screen.delay(5)
# gyorsabb rajzolás
screen.delay()
# 5 - jelenlegi késleltetési idő: 5 ms
```

tracer() | tracer(n) | tracer(n, idő)

Az animáció ki-/bekapcsolása és gyorsítására, paraméter nélkül a képernyőfrissítés sebességének lekérdezésére szolgál. Az `idő` paraméterben megadott ezredmásodpercenként frissíti az a képernyőt a futás közben, az `n` paraméterben megadott lépés után. A rajzolás gyorsítására használható, bonyolult rajzok esetében.

Példa

```
screen.tracer(8, 25) # csak minden 8. lépés látszik, 25 ms-ként
print(screen.tracer())
# 8 - n értéke
screen.tracer(0) # képernyő frissítés kikapcsolás
screen.update() # manuális képernyőfrissítés
```

update()

Képernyőfrissítés kézzel. Akkor kell használni, ha `tracer(0)`-val kikapcsoltad az automatikus frissítést.

Példa

```
screen.tracer(0) # képernyő frissítés kikapcsolás
screen.update() # manuális képernyőfrissítés
```

Eseménykezelés

listen()

A `listen()` utasítás bekapcsolja a billentyűzet figyelését a turtle ablakban. Enélkül a billentyűesemények nem működnek.

Példa:

```
import turtle
screen = turtle.Screen()
```

```
screen.listen()
```

onkey() | onkeyrelease()

Az `onkey()` vagy `onkeyrelease()` egy billentyű elengedéséhez rendel egy függvényt. Amikor a megadott billentyűt felengedjük, a függvény lefut.

Példa

```
import turtle
t = turtle.Turtle()
screen = turtle.Screen()

def elore():
    t.forward(50)

screen.listen()
screen.onkey(elore, "Up")
# A teknőc akkor mozdul előre, amikor felengedjük a fel nyilat.
```

onkeypress()

Az `onkeypress()` egy billentyű lenyomásához rendel egy függvényt. A függvény már a billentyű lenyomásakor lefut (nem elengedéskor).

Példa

```
import turtle
t = turtle.Turtle()
screen = turtle.Screen()

def balra():
    t.left(30)

screen.listen()
screen.onkeypress(balra, "Left")
# A teknőc már a billentyű lenyomásakor balra fordul.
```

onclick(fun, btn=1, add=None)

A megadott `fun` függvényt hozzárendeli a teknőcön történő kattintáshoz.

- Ha `fun` értéke None, az összes korábbi kattintásesemény törlődik.
- A függvény automatikusan megkapja a kattintás `(x, y)` koordinátáit.

Paraméterek:

- `fun` – (kötelező) egy függvény két paraméterrel `(x, y)`, amelyet a rendszer meghív a kattintás helyének koordinátáival,
- `btn` – (opcionális) az egérgomb száma (alapértelmezett: 1 – bal egérgomb)
- `add` – (opcionális) `True` (új eseménykezelőt ad hozzá) vagy `False` (lecseréli a korábbit)

Példa

```
t = turtle.Turtle()
def fordul(x, y):
    t.left(180)

onclick(fordul)   # kattintásra a teknőc megfordul
onclick(None)   # kattintás esemény törlése
```

onrelease(fun, btn=1, add=None)

A megadott `fun` függvényt akkor hívja meg, amikor elengeded az egérgombot a teknőc felett. Ha `fun = None`, az eseménykezelés megszűnik. Paraméterezése ugyanaz, mint az `onclick(fun, btn=1, add=None)` esetén.

Példa

```
def piros(x, y):
    t.fillcolor("red")
def atlatszo(x, y):
    t.fillcolor("")

t = turtle.Turtle()
t.onclick(piros)       # kattintáskor piros lesz
t.onrelease(atlatszo)  # elengedéskor átlátszó
```

ondrag(fun) | onrelease(fun, btn) | onrelease(fun, btn, add)

A `fv` függvényt akkor hívja meg, amikor az egérgomb lenyomva van, és az egér mozog a teknőc fölött (minden húzási eseményt megelőz egy kattintás a teknőcön). Paraméterezése ugyanaz, mint az `onclick(fun, btn=1, add=None)` esetén.

Példa

```
turtle.ondrag(turtle.goto)
# ha a teknőcre kattintva az egeret lenyomva tartjuk, a teknőc követi
az egérkurzor pozícióját, ha a toll le van rakva, rajzol is
```

`ontimer(fv) | ontimer(fv, idő))`

Egy időzítőt állít be, amely `idő` milliszekundum elteltével meghívja a `fv` függvényt.

Ez az animációk alapja `turtle`-ben: a függvény önmagát újra és újra meghívhatja az `ontimer` segítségével.

Paraméterek:

- `fv` – egy *paraméter nélküli függvény*, amelyet az időzítő hív meg,
- `idő` – (opcionális) egy **0-nál nagyobb vagy egyenlő szám**, ez az idő ezredmásodpercben (alapértelmezett: 0).

Példa

```
screen = turtle.Screen()
t = turtle. Turtle()

def mozog():
    if fut: # ha a fut változó értéke igaz
        t.fd(50) # a teknőc előremegy 50-et
        t.lt(60) # a teknőc elfordul 60 fokot
        screen.ontimer(mozog, 250) # az ontimer esemény 250 ms múlva
újra meghívja a mozog függvényt
fut = True
mozog()   # elindítja a teknőc mozgását
fut = False # leállítja a teknőc mozgását
```

`mainloop() | done()`

Elindítja az **eseménykezelő ciklust**, vagyis meghívja a `Tkinter mainloop()` függvényét.

Ez **kötelezően a turtle grafikus program utolsó utasítása** kell legyen.

Nem szabad használni, ha a programot **IDLE-ben `-n` módban (No subprocess)** futtatjuk, vagyis amikor a `turtle` grafikát **interaktívan** használjuk.

Python vezérlő utasítások

for

A `for elem in sorozat` **vezérlési szerkezet** egy sorozat elemein való végigiterálásra szolgál. A sorozat lehet lista, tuple, szótár, halmaz,karakterlánc, vagy más iterálható objektum.

A for ciklus működése eltér a más programozási nyelvekben megszokott `for` kulcsszótól, mivel nem indexalapú, hanem inkább egy iterátoron alapuló működést valósít meg (`for-each`), amely az objektumorientált programozási nyelvekre jellemző.

A **for ciklus** segítségével egy utasítássorozatot hajthatunk végre a sorozat minden egyes elemére pontosan egyszer.

Példa

```
# szöveges lista elemeinek kiíratás
gyumolcsok = ["alma", "banán", "meggy"]
for gyumolcs in gyumolcsok:
    print(gyumolcs)
# karakterlánc karaktereinek kiíratása
for betu in "almafa":
    print(betu)
# számlálós ciklus megvalósítása: egész számok kiíratása 1-től 10-ig
for i in range(10):
    print(i)
```

while

A **while ciklus** egy feltételvezérelt ciklus, amely addig hajt végre egy utasítássorozatot, amíg a megadott logikai feltétel igaz.

A ciklus feltétele minden egyes iteráció előtt kiértékelésre kerül. Ha a feltétel hamissá válik, a ciklus végrehajtása megszakad. Fontos, hogy a cikluson belül gondoskodjunk a feltétel megváltoztatásáról, ellenkező esetben végtelen ciklus jöhet létre.

Példa

```
# Számok kiíratása 1-től 5-ig:
i = 1
while i <= 5:
    print(i)
    i += 1
```

if

Az `if` utasítás feltételes vezérlést valósít meg. Segítségével egy utasítássorozat csak akkor hajtódik végre, ha a megadott feltétel igaz.

Példa

```
x = 10
if x > 0:
    print("A szám pozitív.")
```

`if else`

Az **if–else utasítás** két egymást kizáró végrehajtási ágat biztosít. Ha az `if` feltétele igaz, az `if` blokk hajtódik végre, ellenkező esetben az `else` blokk.

Példa

```
# egy szám előjelének meghatározása
x = -3
if x >= 0:
    print("A szám nem negatív.")
else:
    print("A szám negatív.")
```

`def`

A `def` utasítás függvények definiálására szolgál. A függvény egy újrafelhasználható kódrészlet, amely csak akkor hajtódik végre, amikor meghívjuk.

A függvény paramétereket fogadhat, és visszatérési értéket is adhat.

Példa

```
# Egyszerű függvény definiálása és meghívása:
def koszontes():
    print("Hello, világ!")
koszontes()
# Példa paraméterrel és visszatérési értékkel:
def osszeg(a, b):
    return a + b
eredmeny = osszeg(3, 5)
print(eredmeny)
```

`return`

A `return` utasítás a függvény végrehajtását lezárja, és visszaadja az eredményt a hívó programrésznek.

Példa

```
# Példa paraméterrel és visszatérési értékkel:
def osszeg(a, b):
    return a + b
eredmeny = osszeg(3, 5)
print(eredmeny)
```

Matematikai függvények – math modul

A **math modul** matematikai műveletek és konstansok használatát teszi lehetővé, például gyökvonást, trigonometrikus függvényeket és a π (pi) értékét.

Példa

```
import math
```

---

sqrt(szám)

A `sqrt(szám)` függvény egy szám négyzetgyökét számítja ki. A `szám` paraméter egy nemnegatív szám kell legyen. A függvény eredmény lebegőpontos szám.

Példa

```
import math
eredmeny = math.sqrt(16)
print(eredmeny)
```

sin(x)

A `sin(x)` függvény az `x` szög szinuszát adja meg. Az `x` értéke **radiánban** értendő, nem fokban.

Példa

```
import math
eredmeny = math.sin(math.pi / 2)
print(eredmeny)
```

cos(x)

A `cos(x)` függvény az `x` szög koszinuszát számítja ki. Az `x` értéke **radiánban** van megadva.

Példa

```
import math
eredmeny = math.cos(0)
print(eredmeny)
```

tan(x)

A `tan(x)` függvény az `x` szög tangensét adja meg. Az `x` paraméter radiánban értendő. Bizonyos értékeknél (pl. π/2) a tangens nem értelmezett.

Példa

```
import math
eredmeny = math.tan(math.pi / 4)
print(eredmeny)
```

pi

A `pi` egy matematikai konstans, amely a π (pi) értékét tartalmazza, amelynek értéke megközelítőleg: 3.141592653589793.

Példa

```
import math
print(math.pi)
```

radians(szög_fok)

Ha fokban megadott szögekkel dolgozol, először radiánná kell alakítani őket a `math.radians()` függvénnyel.

Példa

```
import math
szog_fok = 30
szog_radian = math.radians(szog_fok)
print(math.sin(szog_radian))
```