Változók

2025

1 Változók létrehozása és használata

1.1 Változók

Egy változó egy olyan név, amelyhez valamilyen érték rendelhető. A név és az érték közé az '=' értékadó művelti jelet kell kitenni. A programozás során a változóra a neve alapján lehet hivatkozni.

[ ]:
```
nev = "Gábor"
eletkor = 25
magassag = 1.75
hazas = True
```

Névhasználati szabályok: - A változónevek betűket (kis- és nagybetűket egyaránt), számokat és az aláhúzás jelet (_) tartalmazhatnak. Például: '`nev`', '`szam_1`', '`valtozo2`'. - A változónevek nem kezdődhetnek számmal. - A Python érzékeny a kis- és nagybetűkre, tehát a '`Valtozo`' és a '`valtozo`' két különböző változónév. - Törekedjünk olyan változóneveket választani, amelyek leírják a változó tartalmát vagy funkcióját. Ez növeli a kód olvashatóságát. - A Python közösség által elfogadott konvenció, a PEP 8, azt javasolja, hogy változónevek között az aláhúzásjelt használjuk (pl. `valtozo_nev`). - Nem használhatjuk a Pythonban már használt kulcsszavakat.

[1]:
```
import keyword

print("Kulcsszavak: ", *keyword.kwlist)
```

```
Kulcsszavak:  False None True and as assert async await break class continue def
del elif else except finally for from global if import in is lambda nonlocal not
or pass raise return try while with yield
```

1.2 Adattípusok

A Python különböző adattípusokat támogat, például: - szöveg (*string*) - a szöveg tartalmát ' ' (szimpla) vagy " " (dupla idézőjelek) közé kell írni - egész szám (*integer*) - lebegőpontos szám (*float*) - a magyarban megszokott tizedesvessző helyett tizedespontot `'.'` kell használni - komplex szám (*complex*) - olyan lebegőpontos szám, amely képzetes résszel is rendelkezik - logikai érték (*boolean*) - két értéket vehet fel: logikai igaz `True`, vagy logikai hamis `False` - None - ezzel jelöljük hogy még nem kapott értéket - lista, tuple, range, set, dictionarie stb. - ezek gyűjtemények, később tanuljuk

[ ]:
```
# Szöveges változó létrehozása
nev = "Gábor"

# Egész szám változó létrehozása
eletkor = 25

# Lebegőpontos szám változó létrehozása
magassag = 1.75

# Logikai változók létrehozása
hazas = True  # False

# Lista létrehozása
hobbik = ["programozás", "zenehallgatás", "sport"]
```

1.3 Változó értékének kiíratása

A `print()` függvény használható a változók, értékek vagy kifejezések kiíratására a konzolra. (A változónak már ismertnek kell lennie a kiíratás előtt.)
Van két opcionális paramétere: - `sep` paraméter: Meghatározza, hogy az értékeket hogyan választja el a kimeneten. Alapértelmezett értéke a szóköz. - `end` paraméter: Meghatározza, hogy a print függvény milyen karakterrel zárja le a kimenetet. Alapértelmezett értéke egy új sor karakter (`"\n"`)

[ ]:
```
# Egyszerű változó értékének a kiíratása (írja ki a nevét)
print(nev)

# Kifejezés kiíratása (számolja ki (2 + (2*5) -3) )
print(2 + 2 * 5 - 3)

# Több változó kiíratása ( név + életkor)
print(nev, eletkor)

# Érték és szöveg kiíratása együtt ("Név:" + név)
print("Név:", nev)
print("Életkor:", eletkor)
print("Magasság:", magassag)
print("Házas:", hazas)
print("Hobbik:", hobbik)

# Szöveg kiíratása behelyettesítéssel
print("Nevem: {0}, Életkorom: {1}".format(nev, eletkor))

# f-string használata (Python 3.6 vagy újabb verziókban)
print(f"Nevem: {nev}, Életkorom: {eletkor}")

# opcionális paraméterek az elválasztásra és a sorvégre
print("Nevem:", nev, "Életkorom:", eletkor, sep="|", end="év\n")
```

1.4 Dinamikusan változó típusok

Pythonban a változók dinamikus típusúak, ami azt jelenti, hogy a változók típusát nem kell előre meghatározni, és a típus futás közben (meg)változhat. Ugyanaz a változó más típussal rendelkezhet a program futása során. Ez egyszerűsíti a kódot és növeli a rugalmasságot, de ugyanakkor fontos figyelmet fordítani a típusokra, hogy ne vezessenek hibákhoz. A változó aktuális típusát a `type()` függvény segítségével lehet lekérdezni.

[ ]:
```python
# Változó típusának ellenőrzése
print(type(nev))
print(type(eletkor))
print(type(magassag))
print(type(hazas))
print(type(hobbik))

# Dinamikus típusváltás
magassag = 175  # Mostantól int típusú változó lesz
print(type(magassag))

magassag = "alacsony"  # Mostantól str típusú változó lesz
print(type(magassag))

magassag = eletkor  # Mostantól int típusú változó lesz, mert az eletkor is int
print(type(magassag))
```

1.5 Típusok átalakítása

A programozás során gyakran szükséges lehet az adatok típusának megváltoztatása, hogy azokat a kívánt módon lehessen használni vagy más típusú adatokkal tudjon együtt dolgozni.
Az egész számra való átalakítása az `int()`, a lebegőpontos számra való átalakítása a `float()`, a szövegre való átalakítása a `str()` függvény segítségével hajtható végre.
Figyelem: az átalakítás során nem változik meg az átadott változó típusa, hanem új értéket hoz létre.

[ ]:
```python
# Szöveg (string) → Egész szám (integer)
szoveg = "123"
egesz_szam = int(szoveg)  # eredmény: 123

# Szöveg (string) → Lebegőpontos szám (float)
szoveg = "3.14"
lebego_pontos = float(szoveg)  # eredmény: 3.14

# Egész szám (integer) → Lebegőpontos szám (float)
egesz_szam = 42
lebego_pontos = float(egesz_szam)  # eredmény: 42.0

# Lebegőpontos szám (float) → Egész szám (integer)
lebego_pontos = 7.9
```

```
egesz_szam = int(lebego_pontos)  # eredmény: 7

# Egész szám (integer) → Szöveg (string)
egesz_szam = 123
szoveg = str(egesz_szam)  # eredmény: "123"

# Lebegőpontos szám (float) → Szöveg (string)
lebego_pontos = 3.14
szoveg = str(lebego_pontos)  # eredmény: "3.14"
```

A logikai típusra való átalakításnál `bool()`, figyelni kell arra, hogy a `0` egész szám és a `0.0` lebegőpontos szám logikai értéke `False` lesz, minden más szám esetén `True`-t kapunk.
Szöveg konvertálása esetén, az üres `""` szöveg `False` értéket jelent, míg nem üres szöveget átalakítva `True`-t kapunk. (még akkor is, ha a szöveg tartalma "False")

típus	hamis	igaz
logikai	False	True
egész	0	nem 0 (pl. 5, -3)
lebegőpontos	0.0	nem 0.0 (pl. 3.1, -5.4)
szöveg	" "	nem üres (pl. "Helló")

[ ]:
```
# Egész szám (integer) → Logikai (bool)
nulla = 0
nem_nulla = 42

nulla_bool = bool(nulla)  # eredmény: False
nem_nulla_bool = bool(nem_nulla)  # eredmény: True


# Szöveg (string) → Logikai (bool)
ures_sztring = ""
nem_ures_sztring = "Nem üres"

ures_bool = bool(ures_sztring)  # eredmény: False
nem_ures_bool = bool(nem_ures_sztring)  # eredmény: True


# Logikai (bool) → Szöveg (string)
igaz_bool = True
hamis_bool = False

igaz_sztring = str(igaz_bool)  # eredmény: "True"
hamis_sztring = str(hamis_bool)  # eredmény: "False"
```

1.6 Műveletek változókkal

1.6.1 Értékadás

A leggyakrabban alkalmazott művelet az értékadás, amely az = értékadó operátorral történik. Ha több változónak szeretnénk értéket adni, akkor nem szükséges azokat külön sorban megadni, egy sorban is elvégezhetjük. Fontos, hogy a bal oldalon és a jobb oldalon azonos számú elemet kell megadni.

[ ]:
```
x, y, z = 10, 3.5, 5
```

ezzel a megadással könnyedén felcserélhető két érték:

[ ]:
```
a = 10
b = 20

# Csere hagyományos módszerrel
tmp = a
a = b
b = tmp

# Csere python-ban
a, b = b, a
```

Ha több változó is ugyanazt az értéket veszi fel, akkor alkalmazhatjuk a következő megadást:

[ ]:
```
x = y = z = 1
```

1.6.2 Matematikai műveletek (egész és lebegőpontos adatokkal)

[ ]:
```
# Összeadás (5+3)
osszeg = 5 + 3
print("Összeadás:", osszeg)

# Kivonás (8-2)
kulonbseg = 8 - 2
print("Kivonás:", kulonbseg)

# Szorzás (6*4)
szorzat = 6 * 4
print("Szorzás:", szorzat)

# Osztás (eredmény mindig lebegőpontos szám lesz) (10 /3)
hanyados = 10 / 3
print("Osztás:", hanyados)

# Egész osztás (10/3 egész része)
egesz_osztas = 10 // 3
print("Egész osztás:", egesz_osztas)
```

```
# Maradék (modulus) (10/3 maradéka)
maradek = 10 % 3
print("Maradék (modulus):", maradek)

# Hatványozás (2^10)
hatvany = 2**4
print("Hatványozás:", hatvany)

# Abszolút érték ( |-7| )
abszolut_ertek = abs(-7)
print("Abszolút érték:", abszolut_ertek)

# Kerekítés (pi, 2 tizedesjegyre)
kerekitett = round(3.14159, 2)
print("Kerekítés:", kerekitett)
```

1.6.3 Összevont műveletek

A metematikai műveletek esetén (+,-,*,/,//,%,**), ha úgy adunk értéket a változónak, hogy annak aktuális értékét fel kell használni (pl.: `a = a + 2`), akkor az értékadás és a matematikai művelet összevonható.

Az *összevont alakot* így használjuk: `a = a + 2` → `a += 2` - `a = a - 5` → `a -= 5` - …

1.6.4 Matematikai modul

Pythonban több beépített matematikai függvény is található a `math` modulban. Ehhez először importálnod kell a `math` modult.

[ ]:
```
# math modul importálása
import math

# Gyökvonás (16 gyöke)
gyok = math.sqrt(16)
print("Gyökvonás:", gyok)

# Hatványozás (2^3/4)
hatvany = math.pow(2, 3)
print("Hatványozás:", hatvany)

# Logaritmus (alap 10) (log(100))
log10 = math.log10(100)
print("Logaritmus (alap 10):", log10)

# Logaritmus (természetes alapú) (ln(25))
ln = math.log(25)
```

```
print("Logaritmus (természetes alapú):", ln)

# Exponenciális függvény (az előző ln változó exponenciálisa)
exp = math.exp(ln)
print("Exponenciális függvény:", exp)  # kerekítési hiba van

# Pi értéke (3.141592653589793)
pi = math.pi
print("Pi értéke:", pi)

# Trigonometrikus függvények (radiánban) (sin(30fok), cos(60fok), tang(45fok)
sinus = math.sin(30 * (pi / 180))  # 30 fok
cosinus = math.cos(math.radians(60))
tangens = math.tan(math.radians(45))

print("Szinusz:", sinus)
print("Koszinusz:", cosinus)
print("Tangens:", tangens)

# Abszolút érték ( |-7.5| )
abszolut_ertek = math.fabs(-7.5)
print("Abszolút érték:", abszolut_ertek)

# Kerekítés felfelé  (4.3 felfelé kerekítése)
felfele_kerekitett = math.ceil(4.3)
print("Kerekítés felfelé:", felfele_kerekitett)

# Kerekítés lefelé  (4.6 lefelé kerekítése)
lefele_kerekitett = math.floor(4.9)
print("Kerekítés lefelé:", lefele_kerekitett)
```

Létezik komplex számok kezelésére alkalmas `cmath` modul is. A képzetes rész jele a `j`. A tényező és a `j` karakter közé nem kerüljön semmi.

[ ]:
```
# Példa komplex számokra
import cmath

x = 3 + 4j
y = cmath.sqrt(-1)  # 1j

print(x * y)
print(type(x))
```

1.7 Műveletek kiértékelésének sorrendje

A műveleteknek van egy végrehajtási sorrendje (precedenciája), amelyet be kell tartaniuk a műveletek megfelelő elvégzése érdekében. A kifejezések értelmezése során a Python a következő sorrendet alkalmazza: 1. Zárójelek: Legmagasabb precedencia a zárójeleknek van. Ha zárójelek

vannak, akkor a kifejezésük lesz először kiértékelve. 2. Hatványozás: Az ** operátor a következő precedenciával rendelkezik, tehát a hatványozást először végzi el. 3. Előjelek és bitenkénti invertálás: Előjelek (például - és +) következnek. Ezek az operátorok egy változó előtti előjelet változtatják meg. 4. Szorzás, Osztás, Maradék: Szorzás (*), osztás (/), és a maradék (%) operátoroknak ugyanaz a precedenciájuk. Balról jobbra történik a kiértékelés. 5. Összeadás, Kivonás: Az összeadás (+) és kivonás (-) operátoroknak ugyanaz a precedenciájuk. Szintén balról jobbra történik a kiértékelés. 6. Biteltoló műveletek: Egész szám bitjeinek eltolását végző műveletek 7. Bitenkénti ÉS: Két egész érték bintenkénti ÉS művelete 8. Bitenkénti kizáróvagy XOR: Két egész érték bitenkénti XOR művelete 9. Bitenkénti VAGY: Két egész érték bintenkénti VAGY művelete 10. Összehasonlító operátorok: Az összehasonlító operátoroknak (például <, >, <=, >=, ==, !=) közösen van egy precedenciájuk. 11. Logikai NEM: Az not logikai operátornak magasabb precedenciája van, mint az and operátornak. 12. Logikai ÉS: Az and logikai operátornak magasabb precedenciája van, mint az or operátornak. 13. Logikai VAGY: Az or logikai operátor a legkisebb precedenciával rendelkezik a logikai operátorok között.

Műveleti jelek	Funkciók
()	Zárójel
**	Hatványozás
+x -x ~x	Plusz és mínusz előjelek, bitenkénti invertálás
* / // %	Szorzás, Osztás, Egész osztás, Maradék
+ -	Összeadás és kivonás
« »	Bit eltolása balra, jobbra
&	Bitenkénti ÉS
^	Bitenkénti kizáróvagy XOR
|	Bitenkénti VAGY
== != > >= < <= is not is in not	Összehasonlító, Objektumazonosság ellenőrző és Tagságot
in	vizsgáló műveletek
not	Logikai tagadás NEM
and	Logikai ÉS
or	Logikai VAGY

Ha bizonytalanok vagyunk a kiértékelés sorrendjében, akkor használjunk zárójeleket, mert az abban található kifejezések értékelődnek ki először.

[ ]:
```
# 2 ^ 3 ^ 2
print(2**3**2)  # 2^9 = 512
print((2**3) ** 2)  # 8^2 = 64

print(2 + 5 * 3)  # 2 + 15 = 17
print((2 + 5) * 3)  # 7 * 3 = 21
```

1.8 Felhasználói bemenet fogadása

A `input()` függvényt használható a felhasználói bemenet fogadására Pythonban.

[ ]:
```
# Felhasználói bemenet kérése
nev = input("Kérem, adja meg a nevét: ")
```

```
# Az input() mindig szövegként értelmezi a beírt értéket
print("Üdv, " + nev + "!")
```

Fontos megjegyezni, hogy az `input()` függvény által visszaadott érték mindig egy szöveg lesz. Ha számokkal szeretnél dolgozni, akkor át kell alakítanod őket megfelelő típusra (pl., `int()` vagy `float()` használatával).

[ ]:
```
# Szám bekérése és használata
eletkor = input("Kérem, adja meg az életkorát: ")
eletkor = int(eletkor)  # Az input() által kapott szöveget integer típussá␣
 ↪alakítjuk

# Most már integer típussal dolgozhatunk
szuletesi_ev = 2025 - eletkor
print("Születési év:", szuletesi_ev)
```

2 Gyakorló feladatok

1. Feladat

   Egy üzletben nagy leárazások vannak. Kinéztünk egy kabátot, a címkéjén az ár 38 990 Ft. A kabátra 20%-os árengedményt adnak, amelyet a pénztárnál fognak érvényesíteni. Mennyit kell fizetni a kabátért? (csak egészre kerekített Ft-ot tudunk kifizetni)

[ ]:
```
eredeti_ar = 38990
kedvezmeny = 0.20
uj_ar = eredeti_ar * (1 - kedvezmeny)
print(round(uj_ar), "Ft a kedvezményes ár.")
```

2. Feladat

   Konzolról kérje be egy kör átmérőjét. Írja ki a kör kerületét és területét 3 tizedesjegy pontossággal.

[ ]:
```
import math


atmero = float(input("Kérem adja meg a kör átmérőjét"))

kerulet = atmero * math.pi

terulet = atmero**2 * math.pi / 4

print(f"Kerület: {kerulet:.3f}")

print(f"Terület: {terulet:.3f}")
```

3. Feladat

Konzolban adjon meg egy hőmérséklet értéket °C-ban. Számítsa át Fahreinheit-be és írja ki 1 tizedesjegy pontossággal.
$y = (x \cdot \frac{9}{5}) + 32$

[ ]:
```
celsius = float(input("Kérem, adja meg a hőmérsékletet Celsius fokban: "))
fahrenheit = (celsius * 9 / 5) + 32
print(f"A hőmérséklet {celsius:.1f} Celsius fok, ami {fahrenheit:.1f}␣
 ↪Fahrenheit fok.")
```

4. Feladat

Adja meg egy derékszögű háromszög egyik befogóját és az átfogót. Határozza meg a harmadik oldalhosszt.

[ ]:
```
import math


a = float(input("Kérem, adja meg az egyik befogót: "))

c = float(input("Kérem, adja meg az átfogót: "))

b = math.sqrt(c**2 - a**2)

print(f"A derékszögű háromszög harmadik oldala: {b:.2f}")
```

5. Feladat

Készítsen egy programot, amely kiszámolja egy I profil kerületét, területét és másodrendű nyomatékait!
https://hu.wikipedia.org/wiki/M%C3%A1sodrend%C5%B1_nyomat%C3%A9k

[ ]:
```
# méretek mm-ben
b = 60
h = 80
tw = 20
h1 = 70

Iy = (h1 * tw**3) / 12 + ((h - h1) / 2 * b**3) / 12 * 2
Ix = ((b * h**3) - 2 * (b - tw) / 2 * h1**3) / 12

kerulet = 2 * b + 2 * h + (b - tw) * 2
terulet = h * b - 2 * (h1 * (b - tw) / 2)

print(f"Kerület: {kerulet} mm")
print(f"Terület: {terulet} mm2")
print(f"Ix: {Iy} mm4")
print(f"Iy: {Ix} mm4")
```

6. Feladat

Írjon egy programot, ami a kiindulásul fokokban, percekben és másodpercekben megadott szögeket radiánba számolja át.

[ ]:
```python
import math

fok, perc, masodperc = 3, 15, 55

fokba = fok
fokba += perc / 60
fokba += masodperc / 60 / 60
rad = math.radians(fokba)
print(f"{fok}°{perc}'{masodperc}'' = {fokba:.3f}° = {rad:.3f} rad")
```