
Mainīgie, to datu tipi, pamatdarbības darbam ar tiem programmēšanas valodā Python

levads

Python programmēšanas valoda ir plaši izmantota un populāra tās vienkāršības dēļ. Šajā materiālā ir aprakstīti pamati darbam ar mainīgajiem, datu tipi un pamatdarbības, ko var veikt ar šiem mainīgajiem.

Mainīgie (1)

Mainīgie ir “kastes”,
kurās tiek glabāti dati.
Python mainīgie tiek
izveidoti brīdī, kad tiem
tiek piešķirta vērtība.

**Deklarēšana un vērtības
piešķiršanas piemērs:**

```
1  a = 5
2  b = True
3  c = "Hello, world!"
```

Python programmēšanas valodā,
deklarējot mainīgo, nav jāraksta tā datu tips

Mainīgie (2)

Mainīgajam var būt īss nosaukums (x, y) vai aprakstošāks nosaukums (punktiKopa, dzimsanasGads). Python mainīgo lielumu noteikumi:

- Mainīgā nosaukumam jā sākas ar burtu vai pasvītrojuma rakstzīmi, nevar sākties ar skaitli
- Mainīgā nosaukumā var būt tikai burtciparu rakstzīmes un pasvītrojumi
- Mainīgo nosaukumi ir reģistrjutīgi (x un X ir trīs dažādi mainīgie)
- Mainīgā nosaukums nevar būt neviens no Python atslēgvārdiem.

Mainīgie (3)

Derīgs mainīgā nosaukums:

- mansVards= "Gerda"
- mans_vards = "Gerda"
- _mans_vards = "Gerda"
- mansvards = "Gerda"
- MANSVARDS = "Gerda"
- mansvards2 = "Gerda"

Nederīgs mainīgā nosaukums:

- 2mansvards = "Gerda"
- mans-vards = "Gerda"
- mans vards = "Gerda"
- mansVārds = "Gerda"

Vairākvārdu mainīgo nosaukumi

Piemēri:

Camel Case

Katrs vārds, izņemot pirmo, sākas ar lielo burtu: `mansDzimsanasGads`

Pascal Case

Katrs vārds sākas ar lielo burtu: `MansDzimsanasGads`

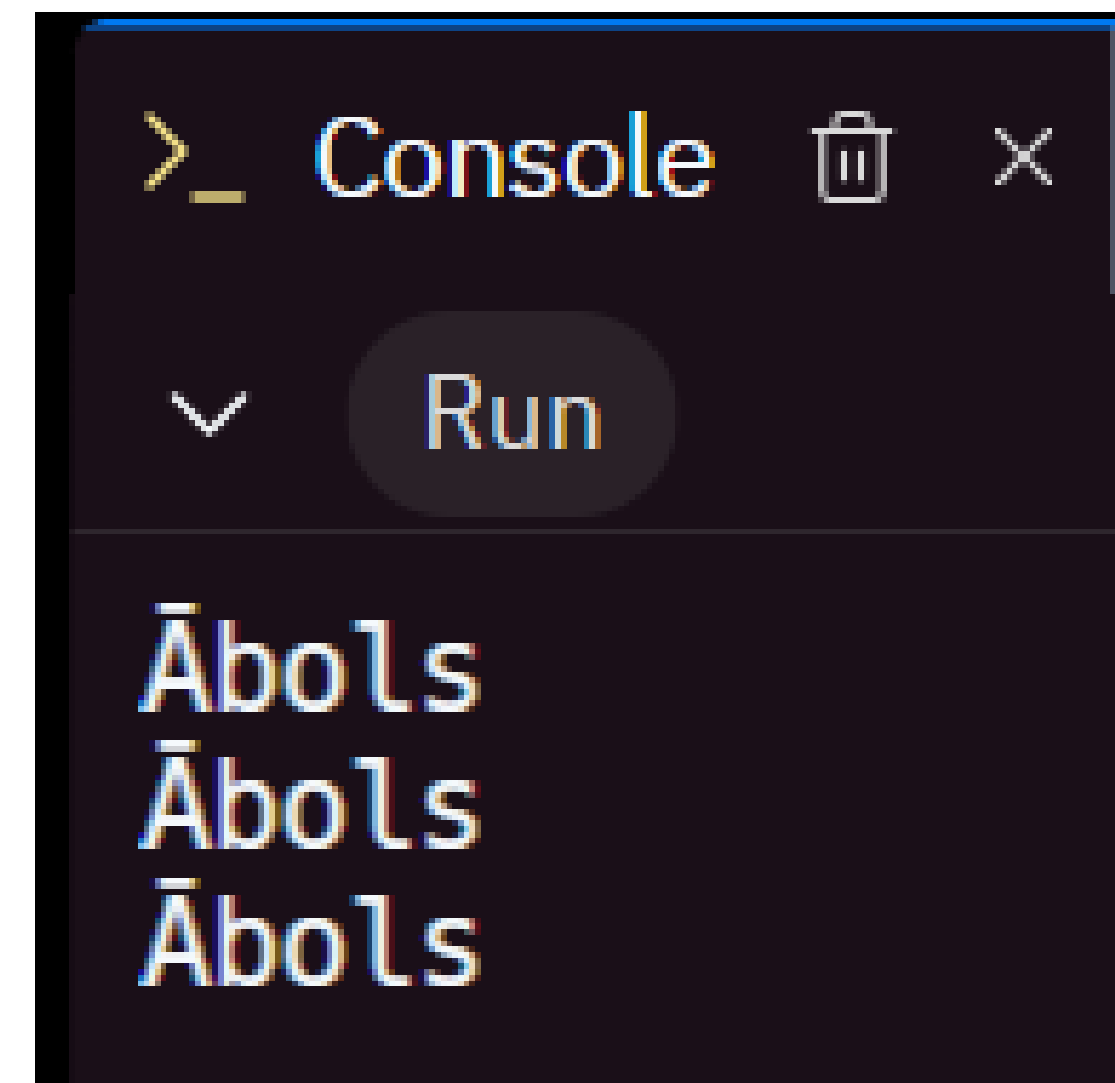
Snake Case

Katru vārdu atdala pasvītrojuma rakstzīme: `mans_dzimsanas_gads`

Viena vērtība vairākiem mainīgajiem

Vienu un to pašu vērtību var piešķirt vairākiem mainīgajiem vienā rindā:

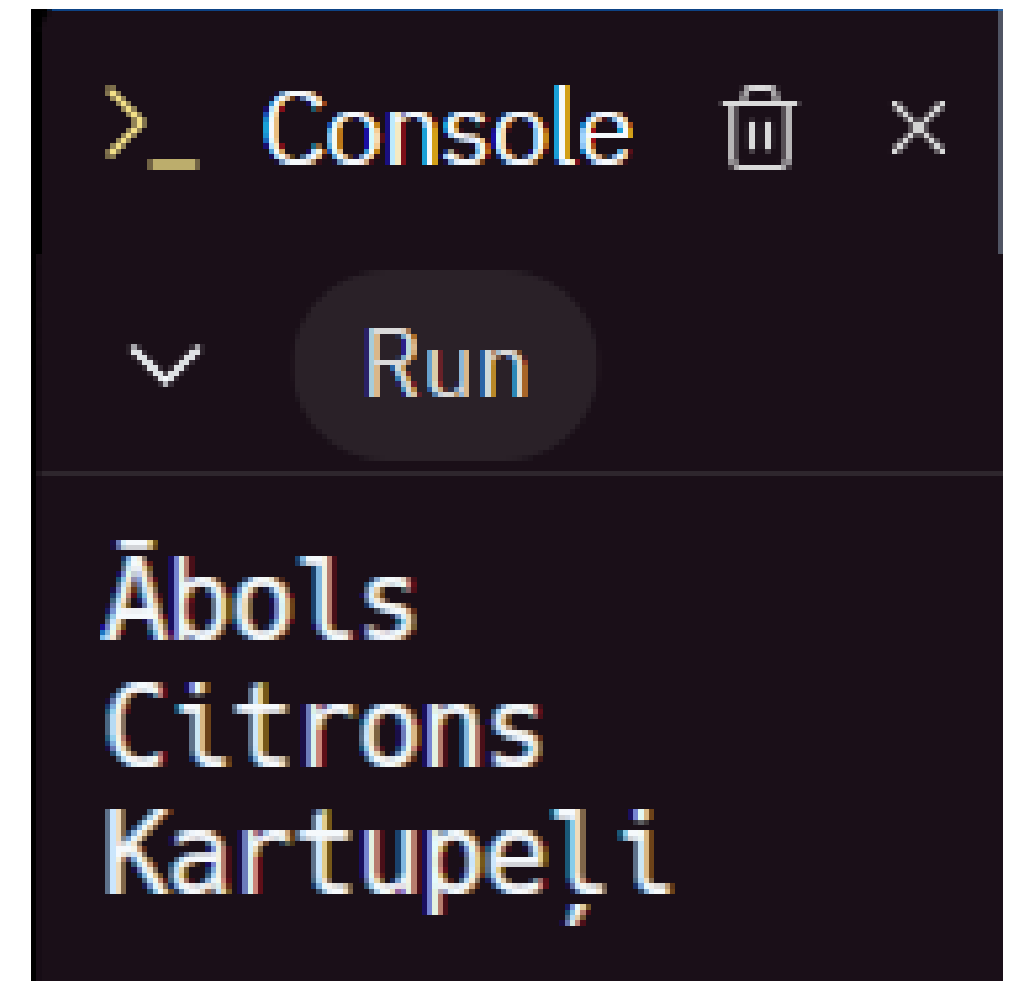
```
a = b = c = "Āboļs"  
print(a)  
print(b)  
print(c)
```



Daudzas vērtības vairākiem mainīgajiem

Python ļauj piešķirt vērtības vairākiem mainīgajiem vienā rindā:

```
1 a, b, c = "Ābols", "Citrons", "Kartupeļi"  
2 print(a)  
3 print(b)  
4 print(c)
```

A screenshot of a Python console window. The window has a title bar with a prompt character '>', the word 'Console', and standard window control icons (minimize, maximize, close). Below the title bar is a 'Run' button. The console output shows three lines of text: 'Ābols', 'Citrons', and 'Kartupeļi', each on a new line.

```
>_ Console [minimize] [maximize] [close]  
v Run  
Ābols  
Citrons  
Kartupeļi
```

Datu tipi

Teksta tips: str

Set tips: set, frozenset

Ciparu tips: int, float, complex

Būla tips: bool

Secības tips: list, tuple, range

None tips: NoneType

Binārie tips: bytes, bytearray, memoryview

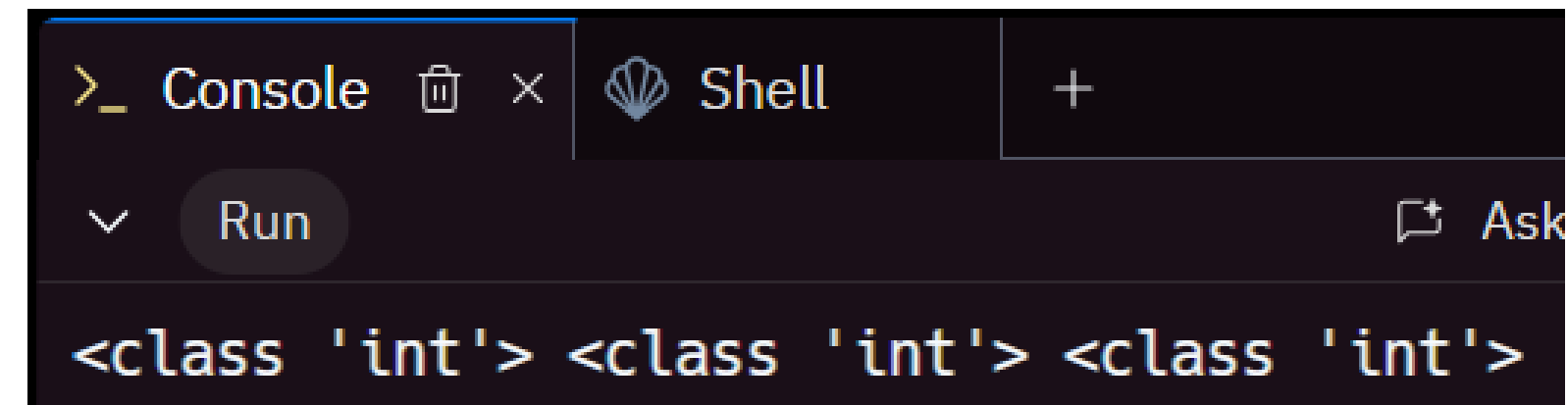
Kartēšanas tips: dict

Tālāk pastāstīšu sīkāk par populārākajiem datu tipiem

Int – vesels skaitlis

Veseli skaitļi ir nulle, pozitīvi un negatīvi skaitļi bez decimāldaļas.

```
1 x, y, z = -1, 0, 1
2 print(type(x), type(y), type(z))
```



The screenshot shows a code editor interface with a dark theme. At the top, there are tabs for 'Console' and 'Shell'. Below the tabs, there is a 'Run' button and an 'Ask' button. The console output displays the result of the Python code: '<class 'int'> <class 'int'> <class 'int'>'. The output is in a monospaced font with syntax highlighting.

```
>_ Console [trash] [x] [shell icon] Shell +
  v Run [Ask]
  <class 'int'> <class 'int'> <class 'int'>
```

Lietošanas piemērs:

```
#Programma lūdz lietotājam ievadīt
#viņa dzimšanas gadu un saglabā šo
#veselo skaitli mainīgajā gads
gads = int(input("Ievadi savu dzimšanas gadu: "))
```

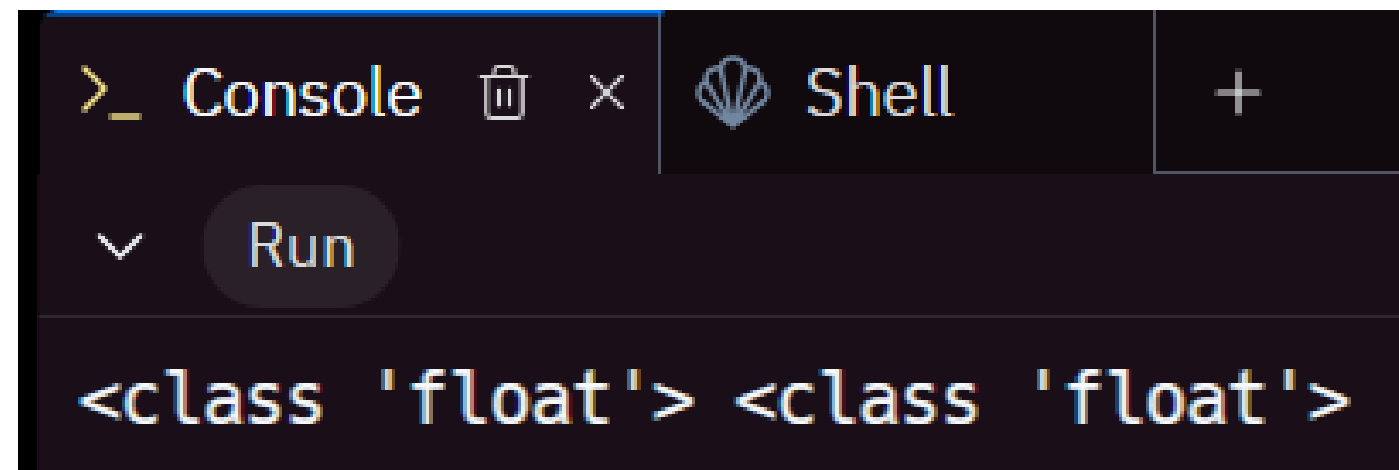
Float – Reāls skaitlis

Reāli skaitļi ir pozitīvs vai negatīvs skaitlis, kas satur decimāldaļu.

```
1 x, y = -1.789, 5.6
2 print(type(x), type(y))
```

Lietošanas piemērs:

```
1 #Programma pieprasa garumu un
2 #saglabā to kā reālu skaitli
3 garums = float(input("Ievadi garumu: "))
```



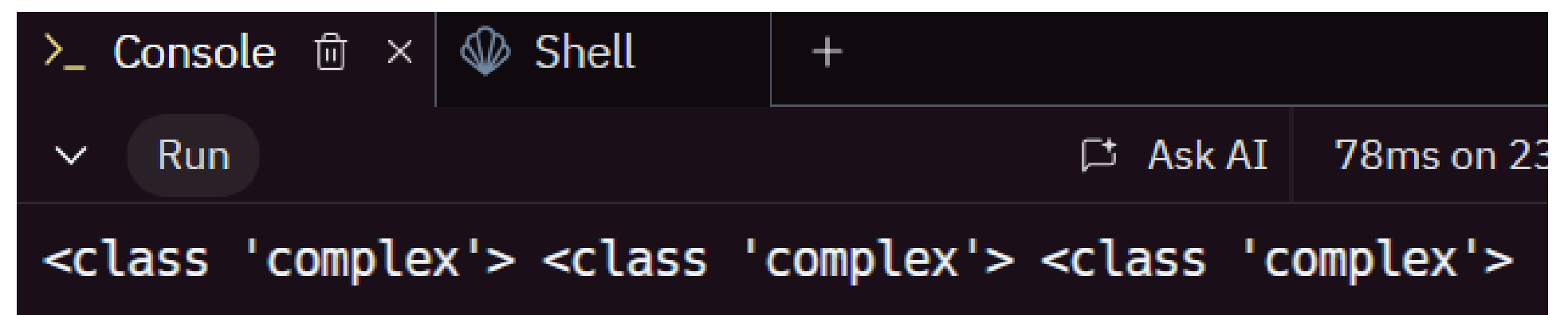
```
>_ Console [trash] [x] [shell icon] Shell [plus]
  v Run
<class 'float'> <class 'float'>
```

! Svarīgi atcerēties, ka i
zmantot komatu
nedrīkst, tikai punktu!

Complex – salikts skaitlis

Salikti skaitļi ir rakstīti ar “j” kā iedomāto daļu.

```
1 x, y, z = 3+5j, 5j, -5j
2 print(type(x), type(y), type(z))
```



```
>_ Console × Shell +
Run Ask AI 78ms on 23
<class 'complex'> <class 'complex'> <class 'complex'>
```

Lietošanas piemērs:

Kompleksos skaitļus izmanto f
izikā un matemātikā

Operatori

+	saskaitīšana	$Z = X + Y$	$Z = 5 + 2$	$Z = 7$
-	atņemšana	$Z = X - Y$	$Z = 5 - 2$	$Z = 3$
*	reizinājums	$Z = X * Y$	$Z = 5 * 2$	$Z = 10$
/	dalīšana	$Z = X / Y$	$Z = 5 / 2$	$Z = 2.5$
%	atlikums	$Z = X \% Y$	$Z = 5 \% 2$	$Z = 1$
**	celt pakāpē	$Z = X ** Y$	$Z = 5 ** 2$	$Z = 25$
//	sadalīt bez atlikuma	$Z = X // Y$	$Z = 5 // 2$	$Z = 2$

Piešķiršanas operatori

Ja mainīgajam jāpievieno/jāatņem noteikta vērtība var izmantot šos divus variantus:

```
1 x = y = 5
2
3 x += 2
4 y = y + 2
5 print(x, y)
```

⌵ Run

7 7

```
1 x = y = 5
2
3 x -= 2
4 y = y - 2
5 print(x, y)
```

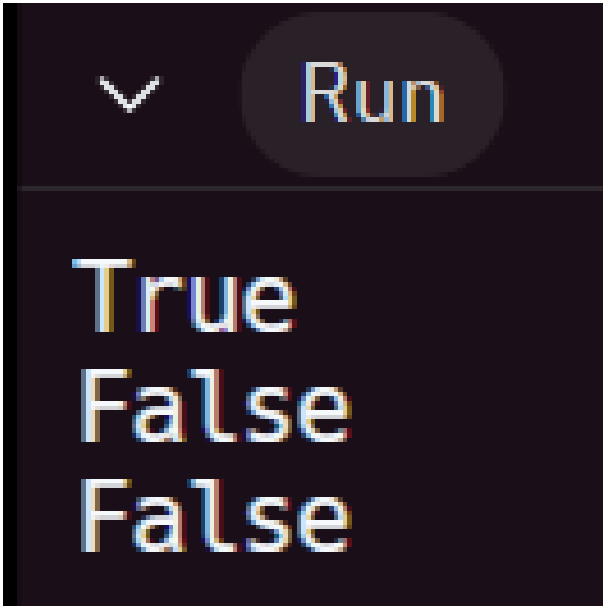
⌵ Run

3 3

Bool

Boļeāni attēlo vienu no divām vērtībām: True vai False.

```
1 print(2 > 1)
2 print(2 == 1)
3 print(2 < 1)
```

A screenshot of a code editor with a dark background. On the left, three lines of Python code are shown: `1 print(2 > 1)`, `2 print(2 == 1)`, and `3 print(2 < 1)`. On the right, there is a 'Run' button and the output of the code: `True`, `False`, and `False` on separate lines.

! Svarīgi atcerēties,
ka Python valodā
True un False obligāti
rakstāma ar lielo burtu!

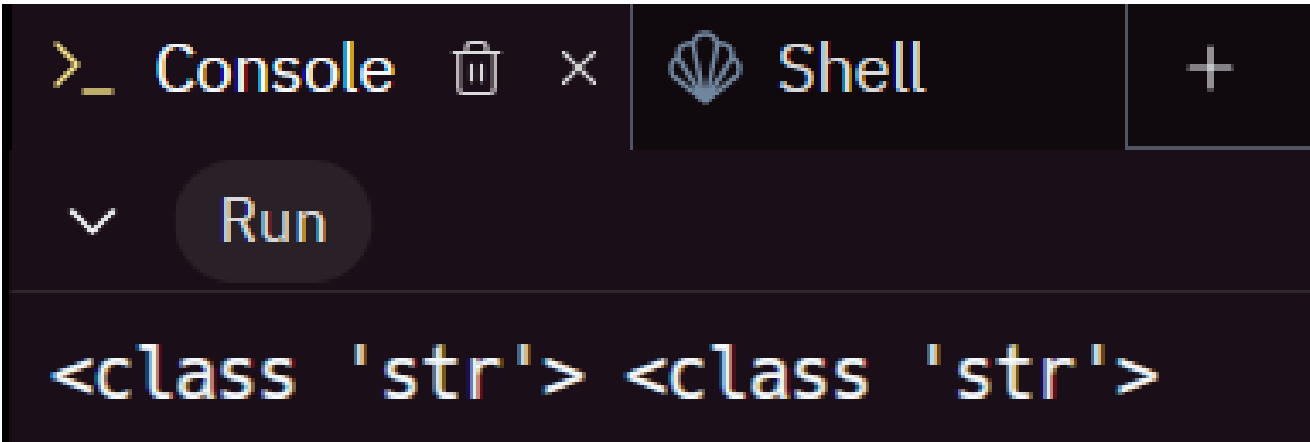
True tiek izmantots, ja darbība ir patiesa, False, kad nav.

Salīdzināšanas operatori: `==` un `!=`

Str – Teksta virkne

String ir teksts, kas Python ir vai nu vienkāršā, vai dubultpēdīnā.

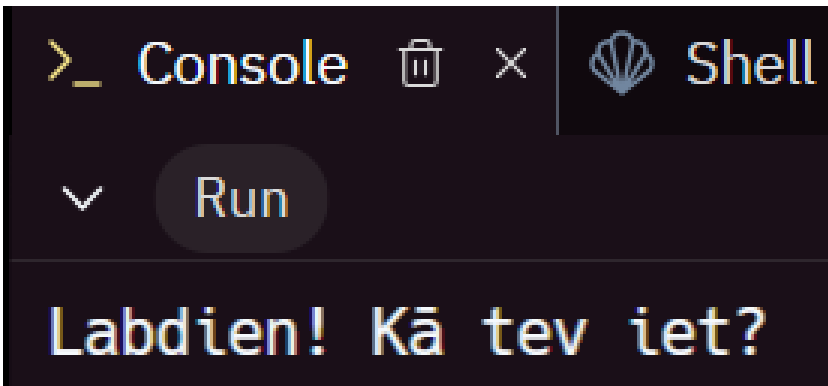
```
1 x, y = "Sveiki", 'Labdien'  
2 print(type(x), type(y))
```



The screenshot shows a terminal window with tabs for 'Console' and 'Shell'. A 'Run' button is visible. The output of the code is displayed as two instances of the string class: `<class 'str'> <class 'str'>`.

Lietošanas piemērs:

```
1 x = "Labdien!"  
2 y = 'Kā tev iet?'  
3 print(x, y)
```



The screenshot shows a terminal window with tabs for 'Console' and 'Shell'. A 'Run' button is visible. The output of the code is displayed as the concatenated strings: `Labdien! Kā tev iet?`.

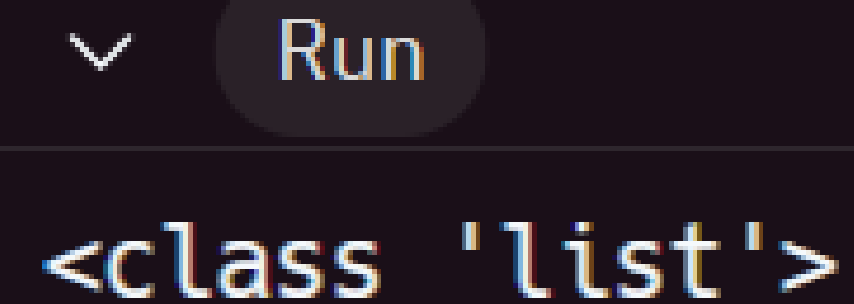


Vienā mainīgajā nevar kombinēt dažādas pēdīņas (`x = 'Sveiki'`)

List – Saraksts

Saraksti tiek izmantoti, lai glabātu vairākus vienumus vienā mainīgajā.

```
1 saraksts = [1, 2, 3, 4]
2 print(type(saraksts))
```

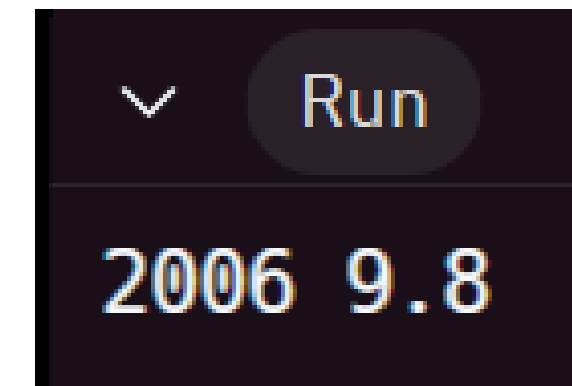


A screenshot of a Python REPL interface. It features a dark background with a light blue 'Run' button and a downward arrow icon. Below the button, the output '<class 'list'>' is displayed in a light blue monospace font.

Vienā sarakstā var būt dažādu tipu dati.

Un piemērs, kā vienumus var izņemt no saraksta.

```
1 saraksts = ["Gerda", 2006, True, 9.8]
2 print(saraksts[1], saraksts[-1])
```



A screenshot of a Python REPL interface. It features a dark background with a light blue 'Run' button and a downward arrow icon. Below the button, the output '2006 9.8' is displayed in a light blue monospace font.

Turple – Kortežs

- Kortežs ir sakārtota un nemainīga kolekcija.
- Turple pieļauj vērtību dublikātus.

```
1 kortezs = ("Gerda", 2006, True, 9.8)
2 print(type(kortezs))
```



Run

```
<class 'tuple'>
```

```
1 tuple = ("apple",)
2 tuple2 = ("apple")
3 print(type(tuple), type(tuple2))
```



Run

```
<class 'tuple'> <class 'str'>
```

Ja kortežam jābūt tikai vienai vērtībai, tad pēc tā jābūt komatam, citādi tas ir str

Set – Kopa

- Kopa ir nesakārtota, nemainīga, neindeksēta kolekcija.
- Kopām nevar būt divi vienumi ar vienādu vērtību.
- Vērtības True un 1 vai False un 0 kopās tiek uzskatītas par vienu un to pašu vērtību, un tās tiek uzskatītas par dublikātiem.

```
1 kopa = {1, 2, 3, 4}
2 print(type(kopa))
```

```
Run
<class 'set'>
```

Dict – Vārdnīca

- Vārdnīcas tiek izmantotas, lai saglabātu datu atslēgas:vērtības pāros.
- Vārdnīca ir sakārtots, maināms un nepieļauj dublikātus.

```
1 x = {"vards": "gerda", "gads": 2004}
2 print(type(x))
```

✓ Run

<class 'dict'>

Metodes

len() - atgriež elementu skaitu objektā.

str

- upper() - atgriež rindas kopiju augšējā reģistrā.
- lower() - atgriež virknes kopiju apakšējā reģistrā.
- capitalize() - atgriež rindas ar lielo pirmo burtu kopiju.
- t() - noņem atstarpes virknes sākumā un beigās.

list

- append(elem) - pievieno elementu saraksta beigām.
- insert(i, elem) - ievieto vienumu norādītajā pozīcijā.
- pop ([i]) - noņem un atgriež indeksu. Ja indekss nav norādīts, tiek paņemts pēdējais elements.
- clear() - no saraksta dzēš visus vienumus.

Metodes

`len()` - atgriež elementu skaitu objektā.

dict

- `keys()` - atgriež visu vārdnīcas atslēgu attēlojumu.
- `values()` - atgriež visu vārdnīcas vērtību attēlojumu.
- `items()` - tiek atgriezts visu vārdnīcas pāru (atslēga, vērtība) attēlojums.

set

- `add(elem)` - pievieno elementu.
- `remove(elem)` - noņem elementu.

tuple

- `count(value)` - tiek atgriezts vērtības gadījumu skaits kortežā.
- `index(value, [start, [stop]])`: tiek atgriezts pirmais vērtības indekss kortežā.

Konkrētā datu tipa iestatīšana

Ja vēlaties norādīt datu tipu,
varat izmantot šādas
konstruktoru funkcijas:

- `x = str("Hello World")`
- `x = int(20)`
- `x = float(20.5)`
- `x = complex(1j)`

- `x = list(("apple", "banana"))`
- `x = tuple(("apple", "banana"))`
- `x = range(6)`
- `x = dict(name="John", age=36)`
- `x = set(("apple", "banana"))`
- `x = frozenset(("apple", "banana"))`
- `x = bool(5)`
- `x = bytes(5)`
- `x = bytearray(5)`
- `x = memoryview(bytes(5))`

Datu tipa iegūšana

Jebkura mainīga datu tipu var iegūt, izmantojot funkcijas:

- `type()`
- `isinstance()`

Piemēri:

```
1 x = 5
2 y = isinstance(x, int)
3 print(y)
```

Konsolē: True

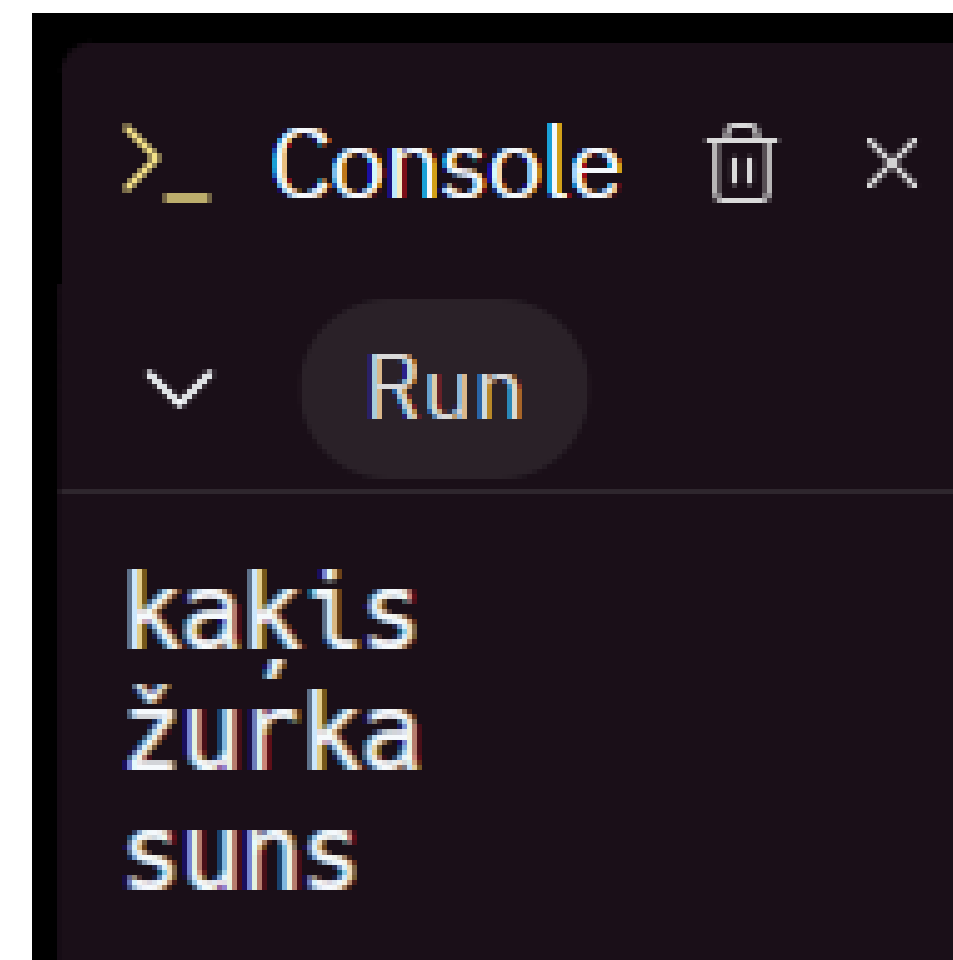
```
1 x = 5
2 y = type(x)
3 print(y)
```

Konsolē: <class 'int'>

Atpakošana

Python ļauj izgūt vērtības no saraksta uz mainīgajiem.

```
dzīvnieki = ["kaķis", "žurka", "suns"]  
x, y, z = dzīvnieki  
print(x)  
print(y)  
print(z)
```



```
>_ Console  
Run  
kaķis  
žurka  
suns
```

Avoti

www.geeksforgeeks.org

python.org

www.w3schools.com

www.learnpython.org

isip.piconepress.com