

Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar
Python
szintaktikai elemek

Dr. Schuster György

2017. november 13.

Bevezető
Változók
Operátorok
Utasítások
Függvények
Modulok

Általános jellemzők
Története
Egy kis szintaxis

Általános jellemzők

Általános jellemzők

Általános célú.

Általános jellemzők

Általános célú.

Általános célú

Bármilyen általános programozási feladatra használható.

Real-time felhasználásra nem ajánlott.

Általános jellemzők

Általános célú.

Imperatív.

Általános jellemzők

Általános célú.

Imperatív.

Imperatív

Utasítások, parancsok, változók használata.
A programozó saját maga írja az algoritmusokat.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Procedurális

A programokat többé - kevésbé független alprogramokra bontja.
Szubrutinok, eljárások használata.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Objektum orientált.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Objektum orientált.

Objektum orientált

Ismeri az OOP tulajdonságait:

- többszörös öröklődés
- operátor overloading
- virtuális függvények.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Objektum orientált.

Multiplatformos.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Objektum orientált.

Multiplatformos.

Multiplatformos

Fut:

- Linux és más UNIX alapú rendszeren
- Windows-on
- DOS-on is.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Objektum orientált.

Multiplatformos.

Interpreteres.

Általános jellemzők

Általános célú.

Imperatív.

Procedurális.

Objektum orientált.

Multiplatformos.

Interpreteres.

Interpreteres

A futtató rendszer a futás során sorról - sorra olvassa a kódot és hajtja végre.
(Vagy JVM kódot gyárt és az futtatja.)

Bevezető
Változók
Operátorok
Utasítások
Függvények
Modulok

Általános jellemzők
Története
Egy kis szintaxis

Története

Története

- **1989** a fejlesztés megkezdése.
(Guido van Rossum
Centrum Wiskunde & Informatica Amsterdam)

Története

- **1989** a fejlesztés megkezdése.
(Guido van Rossum
Centrum Wiskunde & Informatica Amsterdam)
- **1994** comp.lang.python hírcsoport létrejön.

Története

- **1989** a fejlesztés megkezdése.
(Guido van Rossum
Centrum Wiskunde & Informatica Amsterdam)
- **1994** comp.lang.python hírcsoport létrejön.
- **1994** Python 1.0 verzió.

Története

- **1989** a fejlesztés megkezdése.
(Guido van Rossum
Centrum Wiskunde & Informatica Amsterdam)
- **1994** comp.lang.python hírcsoport létrejön.
- **1994** Python 1.0 verzió.
- **2000** Python 2.0 verzió.

Története

- **1989** a fejlesztés megkezdése.
(Guido van Rossum
Centrum Wiskunde & Informatica Amsterdam)
- **1994** comp.lang.python hírcsoport létrejön.
- **1994** Python 1.0 verzió.
- **2000** Python 2.0 verzió.
- **2008** Python 3.0 verzió.

Története

- **1989** a fejlesztés megkezdése.
(Guido van Rossum
Centrum Wiskunde & Informatica Amsterdam)
- **1994** comp.lang.python hírcsoport létrejön.
- **1994** Python 1.0 verzió.
- **2000** Python 2.0 verzió.
- **2008** Python 3.0 verzió.
- **Folyamatos fejlesztés.**

Története

- **1989** a fejlesztés megkezdése.
(Guido van Rossum
Centrum Wiskunde & Informatica Amsterdam)
- **1994** comp.lang.python hírcsoport létrejön.
- **1994** Python 1.0 verzió.
- **2000** Python 2.0 verzió.
- **2008** Python 3.0 verzió.
- **Folyamatos fejlesztés.**
- **Kezdi kiszorítani a hagyományos nyelveket.**

Egy kis szintaxis

Egy kis szintaxis

Érzékeny a kis és nagybetűkre:
VALUE és **value** nem ugyanaz.

Egy kis szintaxis

Érzékeny a kis és nagybetűkre:
VALUE és **value** nem ugyanaz.

Egy kifejezést egy sorvég karakter zár le:
i = 5

Egy kis szintaxis

Érzékeny a kis és nagybetűkre:
VALUE és **value** nem ugyanaz.

Egy kifejezést egy sorvég karakter zár le:
`i = 5`

Egy összefüggő logikai blokkot egy tabulátorral beljebb írnak:
`while (i <= 5) :`
`→ i = i+1`

Bevezető
Változók
Operátorok
Utasítások
Függvények
Modulok

Int, Float és Complex típusok
Bool típus
String típus
List és Tuple típusok
Bytes és bytearray típusok
Set és Frozenset típusok
Dictionaries típus

Általános ismeretek

Általános ismeretek

- 1 A változót nem kell deklarálni.

Általános ismeretek

- 1 A változót nem kell deklarálni.
- 2 Vannak változó típusok.

Általános ismeretek

- 1 A változót nem kell deklarálni.
- 2 Vannak változó típusok.
- 3 A típus a változó létrejöttékor definiálásra kerül.

Általános ismeretek

- 1 A változót nem kell deklarálni.
- 2 Vannak változó típusok.
- 3 A típus a változó létrejöttékor definiálásra kerül.
- 4 A definiált típus mindig az adott értékre „próbál” optimális lenni.

Általános ismeretek

- 1 A változót nem kell deklarálni.
- 2 Vannak változó típusok.
- 3 A típus a változó létrejöttkor definiálásra kerül.
- 4 A definiált típus mindig az adott értékre „próbál” optimális lenni.
- 5 A változó neve:
 - a karakter halmaz [_a-zA-Z0-9],
 - a változó nevének első karaktere [_a-zA-Z].

Általános ismeretek

- 1 A változót nem kell deklarálni.
- 2 Vannak változó típusok.
- 3 A típus a változó létrejöttékor definiálásra kerül.
- 4 A definiált típus mindig az adott értékre „próbál” optimális lenni.
- 5 A változó neve:
a karakter halmaz [_a-zA-Z0-9],
a változó nevének első karaktere [_a-zA-Z].

A tizedespont jelenléte értékadásnál lebegőpontos típust definiál.

Változó típusok (felsorolás)

`int`

`float`

`complex`

`str`

`bool`

`lists`

`bytes`

`bytearray`

`tuple`

`set`

`frozenset`

`dict`

int típus

Tetszőleges méretű egész érték tárolására szolgál.

int típus

Tetszőleges méretű egész érték tárolására szolgál.

```
a = 1
```

int típus

Tetszőleges méretű egész érték tárolására szolgál.

```
a = 1  
a = 10
```

int típus

Tetszőleges méretű egész érték tárolására szolgál.

```
a = 1  
a = 10  
a = 100
```

int típus

Tetszőleges méretű egész érték tárolására szolgál.

```
a = 1  
a = 10  
a = 100  
a = 1000
```

int típus

Tetszőleges méretű egész érték tárolására szolgál.

```
a = 1
a = 10
a = 100
a = 1000
  ⋮
a = 1000000000000000000000000...
  ⋮
```


int típus

Tetszőleges méretű egész érték tárolására szolgál.

```
a = - 1
a = - 10
a = - 100
a = - 1000
  ⋮
a = - 1000000000000000000000000...
  ⋮
```

float típus

Rendszertől függő pontosságú lebegőpontos szám tárolására szolgál.

```
b = 3.1415926535
```

Ez egyértelműen egy lebegőpontos szám, de ...

```
a = 1
```

float típus

Rendszertől függő pontosságú lebegőpontos szám tárolására szolgál.

```
b = 3.1415926535
```

Ez egyértelműen egy lebegőpontos szám, de ...

```
a = 1
```

Ez eddig egy `int`, de

float típus

Rendszertől függő pontosságú lebegőpontos szám tárolására szolgál.

```
b = 3.1415926535
```

Ez egyértelműen egy lebegőpontos szám, de ...

```
a = 1
```

Ez eddig egy `int`, de

```
a = a + 0.1
```

float típus

Rendszertől függő pontosságú lebegőpontos szám tárolására szolgál.

```
b = 3.1415926535
```

Ez egyértelműen egy lebegőpontos szám, de ...

```
a = 1
```

Ez eddig egy `int`, de

```
a = a + 0.1
```

ettől kezdve már `float`.

Átmenet a `int` és `float` típusok között

Átmenet a `int` és `float` típusok között

`int` → `float`

Átmenet a `int` és `float` típusok között

```
int → float  
>>> a = 5
```


Átmenet a `int` és `float` típusok között

`int → float`

```
>>> a = 5
```

```
>>> b = float(a)
```

Átmenet a `int` és `float` típusok között

```
int → float  
>>> a = 5  
>>> b = float(a)  
>>> print(b)  
5.0
```

Átmenet a `int` és `float` típusok között

```
int → float  
>>> a = 5  
>>> b = float(a)  
>>> print(b)  
5.0
```

print függvény

A standard kimenetre ír
formátumozottan is.

Később részletesen
tárgyaljuk.

Átmenet a `int` és `float` típusok között

`int → float`

```
>>> a = 5  
>>> b = float(a)  
>>> print(b)  
5.0
```

`float → int`

Átmenet a `int` és `float` típusok között

`int → float`

```
>>> a = 5
>>> b = float(a)
>>> print(b)
5.0
```

`float → int`

```
>>> a = 5.5
```

Átmenet a `int` és `float` típusok között

`int → float`

```
>>> a = 5
>>> b = float(a)
>>> print(b)
5.0
```

`float → int`

```
>>> a = 5.5
>>> b = int(a)
```

Átmenet a `int` és `float` típusok között

`int → float`

```
>>> a = 5
>>> b = float(a)
>>> print(b)
5.0
```

`float → int`

```
>>> a = 5.5
>>> b = int(a)
>>> print(b)
5
```

Átmenet a `int` és `float` típusok között

`int → float`

```
>>> a = 5
>>> b = float(a)
>>> print(b)
5.0
```

`float → int`

```
>>> a = 5.5
>>> b = int(a)
>>> print(b)
5
```

Az `int()` függvény egészérték függvény.

Tehát **nem kerekít**, csak levágja a törtrészt.

complex típus

`c = 3 + 6j`

complex típus

```
c = 3 + 6j  
>>> print(c)
```

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

```
c = 3.5 + 6.2j
```

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

```
c = 3.5 + 6.2j  
>>> print(c)  
(3.5+6.2j)
```

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

```
c = 3.5 + 6.2j  
>>> print(c)  
(3.5+6.2j)
```

A komplex szám jellemzői:

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

```
c = 3.5 + 6.2j  
>>> print(c)  
(3.5+6.2j)
```

A komplex szám jellemzői:

```
>>> print a.real
```


complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

```
c = 3.5 + 6.2j  
>>> print(c)  
(3.5+6.2j)
```

A komplex szám jellemzői:

```
>>> print a.real  
3
```

```
>>> print a.imag
```

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

```
c = 3.5 + 6.2j  
>>> print(c)  
(3.5+6.2j)
```

A komplex szám jellemzői:

```
>>> print a.real  
3
```

```
>>> print a.imag  
6
```

```
>>> print abs(a)
```

complex típus

```
c = 3 + 6j  
>>> print(c)  
(3+6j)
```

vagy

```
c = 3.5 + 6.2j  
>>> print(c)  
(3.5+6.2j)
```

A komplex szám jellemzői:

```
>>> print a.real  
3
```

```
>>> print a.imag  
6
```

```
>>> print abs(a)  
6.70820
```

bool típus

Csak két értéket vehet fel ezek:

bool típus

Csak két értéket vehet fel ezek:

True igaz érték,
False hamis érték.

bool típus

Csak két értéket vehet fel ezek:

True igaz érték,
False hamis érték.

Az értékadás hasonló.

bool típus

Csak két értéket vehet fel ezek:

True igaz érték,
False hamis érték.

Az értékadás hasonló.

```
>>> a=True  
>>> print a  
True
```

bool típus

Csak két értéket vehet fel ezek:

True igaz érték,
False hamis érték.

Az értékadás hasonló.

```
>>> a=True  
>>> print a  
True
```

illetve

```
>>> a=False  
>>> print a  
False
```


String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

```
str1="This is a Python string."
```

String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

```
str1="This is a Python string."
```

Második forma:

```
str1='This is a Python string.'
```

String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

```
str1="This is a Python string."
```

Második forma:

```
str1='This is a Python string.'
```

Mi a különbség?

String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

```
str1="This is a Python string."
```

Második forma:

```
str1='This is a Python string.'
```

Mi a különbség? Szinte semmi, de

String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

```
str1="This is a Python string."
```

Második forma:

```
str1='This is a Python string.'
```

Mi a különbség? Szinte semmi, de ha a stringben szerepel ', vagy ", akkor határolónak a másikat használjuk.

String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

```
str1="This is a Python string."
```

Második forma:

```
str1='This is a Python string.'
```

Mi a különbség? Szinte semmi, de ha a stringben szerepel ', vagy ", akkor határolónak a másikat használjuk.

```
>>> str1='This is a "Python" string.'
```

```
>>> print str1
```

```
This is a "Python" string.'
```

String típus

A string egy UTF8 karakterlánc. Két formátum létezik.

Első forma:

```
str1="This is a Python string."
```

Második forma:

```
str1='This is a Python string.'
```

Mi a különbség? Szinte semmi, de ha a stringben szerepel ', vagy ", akkor határolónak a másikat használjuk.

```
>>> str1='This is a "Python" string.'
```

```
>>> print str1
```

```
This is a "Python" string.'
```

```
>>> str2="This isn't a Perl string."
```

```
>>> print str2
```

```
This isn't a Perl string.
```


String

A string karakterei elérhetők.

String

A string karakterei elérhetők.

Egyenként:

String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"
```

String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"  
>>> print str[1]
```

String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"  
>>> print str[1]  
e
```

String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"  
>>> print str[1]  
e
```

Tartomány:

String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"  
>>> print str[1]  
e
```

Tartomány:

```
>>> str="Hello!"
```

String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"  
>>> print str[1]  
e
```

Tartomány:

```
>>> str="Hello!"  
>>> print str[1:3]
```


String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"  
>>> print str[1]  
e
```

Tartomány:

```
>>> str="Hello!"  
>>> print str[1:3]  
el
```

String

A string karakterei elérhetők.

Egyenként:

```
>>> str="Hello!"  
>>> print str[1]  
e
```

Tartomány:

```
>>> str="Hello!"  
>>> print str[1:3]  
el
```

A tartomány esetén a második index a felső határ, azt **már nem** tartalmazza a „szelet”.

String

Vezérlő karakterek (\ karakterek ESC szekvenciák)
elhelyezése a stringben.

String

Vezérlő karakterek (\ karakterek ESC szekvenciák) elhelyezése a stringben.

```
>>> str=r"This is an LF sequence \n"
```

String

Vezérlő karakterek (\ karakterek ESC szekvenciák) elhelyezése a stringben.

```
>>> str=r"This is an LF sequence \n"  
>>> print str  
This is a LF sequence \n
```

String

Vezérlő karakterek (\ karakterek ESC szekvenciák) elhelyezése a stringben.

```
>>> str=r"This is an LF sequence \n"
>>> print str
This is a LF sequence \n
```

ESC szekvenciák:

<code>\n</code>	-	soremelés	<code>\cx</code>	-	Ctrl+x
<code>\r</code>	-	kocsivissza	<code>\C-x</code>	-	Ctrl+x
<code>\t</code>	-	tabulátor	<code>\e</code>	-	escape
<code>\a</code>	-	csengő	<code>\f</code>	-	lapdobás
<code>\v</code>	-	függőleges tabulátor	<code>\M-\C-x</code>	-	Meta+Control+x

list típus

list típus

Klasszikus egydimenziós tömb.

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
```

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]  
>>>print lst1[1]
```

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2
```

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2
>>> lst2=["Humpty", "Dumpty", 95]
>>> print lst2[0]
```

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2
>>> lst2=["Humpty", "Dumpty", 95]
>>> print lst2[0]
Humpty
```

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2
>>> lst2=["Humpty", "Dumpty", 95]
>>> print lst2[0]
Humpty
print lst2
```

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2
>>> lst2=["Humpty", "Dumpty", 95]
>>> print lst2[0]
Humpty

print lst2
["Humpty", "Dumpty", 95]
```

list típus

Klasszikus egydimenziós tömb.

**Bővíteni csak
függvénnyel lehet!**

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2

>>> lst2=["Humpty", "Dumpty", 95]
>>> print lst2[0]
Humpty

print lst2
["Humpty", "Dumpty", 95]
```


list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2

>>> lst2=["Humpty", "Dumpty", 95]
>>> print lst2[0]
Humpty

print lst2
["Humpty", "Dumpty", 95]
```

**Bővíteni csak
függvénnyel lehet!**

A negatív indexelés
lehetséges.

Pl.: öt elemű tömb
esetén.

list típus

Klasszikus egydimenziós tömb.

```
>>>lst1=[1, 2, 3]
>>>print lst1[1]
2
>>> lst2=["Humpty", "Dumpty", 95]
>>> print lst2[0]
Humpty

print lst2
["Humpty", "Dumpty", 95]
```

**Bővíteni csak
függvénnyel lehet!**

A negatív indexelés
lehetséges.

Pl.: öt elemű tömb
esetén.

Oda	Vissza
0	-5
1	-4
2	-3
3	-2
4	-1

tuple típus

Klasszikus egydimenziós tömb, **de ez nem változtatható.**

tuple típus

Klasszikus egydimenziós tömb, **de ez nem változtatható.**

```
>>> tp11=(1, 2, 3)
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tpl1=(1, 2, 3)
>>> print tpl1[1]
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tp11=(1, 2, 3)
>>> print tp11[1]
2
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tp11=(1, 2, 3)
>>> print tp11[1]
2
>>> tp12=("Humpty", "Dumpty", 95)
>>> print tp12[0]
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tp11=(1, 2, 3)
>>> print tp11[1]
2
>>> tp12=("Humpty", "Dumpty", 95)
>>> print tp12[0]
Humpty
```


tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tp11=(1, 2, 3)
>>> print tp11[1]
2
>>> tp12=("Humpty", "Dumpty", 95)
>>> print tp12[0]
Humpty
print tp12
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tp11=(1, 2, 3)
>>> print tp11[1]
2
>>> tp12=("Humpty", "Dumpty", 95)
>>> print tp12[0]
Humpty
print tp12
("Humpty", "Dumpty", 95)
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tp11=(1, 2, 3)
>>> print tp11[1]
2
>>> tp12=("Humpty", "Dumpty", 95)
>>> print tp12[0]
Humpty
print tp12
("Humpty", "Dumpty", 95)
>>> t[1]=3
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tpl1=(1, 2, 3)
>>> print tpl1[1]
2
>>> tpl2=("Humpty", "Dumpty", 95)
>>> print tpl2[0]
Humpty
print tpl2
("Humpty", "Dumpty", 95)
>>> t[1]=3
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not
support item assignment
>>>
```

tuple típus

Klasszikus egydimenziós tömb, de ez nem változtatható.

```
>>> tpl1=(1, 2, 3)
>>> print tpl1[1]
2
>>> tpl2=("Humpty", "Dumpty", 95)
>>> print tpl2[0]
Humpty
print tpl2
("Humpty", "Dumpty", 95)
>>> t[1]=3
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not
support item assignment
>>>
```

Az alapvető különbség a zárójel:

- a lists esetén [],

- a tuple esetén ().

Bevezető
Változók
Operátorok
Utasítások
Függvények
Modulok

Int, Float és Complex típusok
Bool típus
String típus
List és Tuple típusok
Bytes és bytearray típusok
Set és Frozenset típusok
Dictionaries típus

bytes típus

bytes típus

Ez a típus nem változtatható bájt szekvenciát tárol.

bytes típus

Ez a típus nem változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el, **de nem változtatható**.

bytes típus

Ez a típus nem változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el, **de nem változtatható.**

```
>>> bts=bytes(b'\x41\x42')
```

bytes típus

Ez a típus nem változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el, **de nem változtatható**.

```
>>> bts=bytes(b'\x41\x42')  
>>> print bts[1]  
B
```

bytes típus

Ez a típus nem változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el, **de nem változtatható**.

```
>>> bts=bytes(b'\x41\x42')
>>> print bts[1]
B
bts[1]=5
```

bytes típus

Ez a típus nem változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el, **de nem változtatható**.

```
>>> bts=bytes(b'\x41\x42')
>>> print bts[1]
B
bts[1]=5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

bytearray típus

bytearray típus

Ez a típus változtatható bájt szekvenciát tárol.

bytearray típus

Ez a típus változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el és **változtatható**.

bytearray típus

Ez a típus változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el és **változtatható**.

```
>>> bts=bytearray(b'\x41\x42')
```


bytearray típus

Ez a típus változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el és **változtatható**.

```
>>> bts=bytearray(b'\x41\x42')  
>>> print bts[1]  
B
```

bytearray típus

Ez a típus változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el és **változtatható**.

```
>>> bts=bytearray(b'\x41\x42')
>>> print bts[1]
B
bts[1]=5
```

bytearray típus

Ez a típus változtatható bájt szekvenciát tárol.

Egy bájt hozzáférése az index segítségével érhető el és **változtatható**.

```
>>> bts=bytearray(b'\x41\x42')
>>> print bts[1]
B
bts[1]=5
>>> print bts[1]
5
```

set típus

set típus

Rendezetlen halmaz, amely ismeri az idempotetitást.

set típus

Rendezetlen halmaz, amely ismeri az idempotetizációt.

```
>>> a={1,2,3}
```

set típus

Rendezetlen halmaz, amely ismeri az indempotetitást.

```
>>> a={1,2,3}  
>>> print a  
set([1, 2, 3])
```

set típus

Rendezetlen halmaz, amely ismeri az idempotenciát.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
```


set típus

Rendezetlen halmaz, amely ismeri az idempotetitást.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
set([1, 2, 3])
```

set típus

Rendezetlen halmaz, amely ismeri az idempotetitást.

A halmaz **nem indexelhető**, **de bővíthető**.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
set([1, 2, 3])
```

set típus

Rendezetlen halmaz, amely ismeri az idempotetitást.

A halmaz **nem indexelhető**, de **bővíthető**.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
set([1, 2, 3])
```

```
>>> a={1,2,3}
>>> a[1]=3
```

set típus

Rendezetlen halmaz, amely ismeri az idempotenciát.

A halmaz **nem indexelhető**, **de bővíthető**.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
set([1, 2, 3])
```

```
>>> a={1,2,3}
>>> a[1]=3

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not
support item assignment
```

set típus

Rendezetlen halmaz, amely ismeri az idempotenciát.

A halmaz **nem indexelhető**, **de bővíthető**.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
set([1, 2, 3])
```

```
>>> a={1,2,3}
>>> a[1]=3

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not
support item assignment

>>> a={1,2,3}
```

set típus

Rendezetlen halmaz, amely ismeri az idempotenciát.

A halmaz **nem indexelhető**, **de bővíthető**.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
set([1, 2, 3])
```

```
>>> a={1,2,3}
>>> a[1]=3

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not
support item assignment

>>> a={1,2,3}
>>> a.add(4)
```

set típus

Rendezetlen halmaz, amely ismeri az idempotenciát.

A halmaz **nem indexelhető**, de **bővíthető**.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> a={1,2,3,3}
>>> print a
set([1, 2, 3])
```

```
>>> a={1,2,3}
>>> a[1]=3

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not
support item assignment

>>> a={1,2,3}
>>> a.add(4)
>>> a

set([1, 2, 3, 4])
```

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

```
>>> a={1,2,3}  
>>> print a  
set([1, 2, 3])
```

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> b=frozenset(a)
>>> print b
```

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> b=frozenset(a)
>>> print b
frozenset([1, 2, 3])
```

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

```
>>> a={1,2,3}  
>>> print a  
set([1, 2, 3])
```

A halmaz **nem indexelhető** és

```
>>> b=frozenset(a)  
>>> print b  
frozenset([1, 2, 3])
```

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> b=frozenset(a)
>>> print b
frozenset([1, 2, 3])
```

A halmaz **nem indexelhető** és **nem bővíthető**.

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> b=frozenset(a)
>>> print b
frozenset([1, 2, 3])
```

A halmaz **nem indexelhető** és **nem bővíthető**.

```
>>> b.add(4)
```

frozenset típus

Rendezetlen halmaz, amely ismeri az indempotetitást és nem változtatható.

```
>>> a={1,2,3}
>>> print a
set([1, 2, 3])

>>> b=frozenset(a)
>>> print b
frozenset([1, 2, 3])
```

A halmaz **nem indexelhető** és **nem bővíthető**.

```
>>> b.add(4)
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has
no attribute 'add'
```

dictionaries típus

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one', 2:'two', 3:'three'}
```

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one',2:'two',3:'three'}  
>>> d
```

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one',2:'two',3:'three'}  
>>> d  
  
{1:'one', 2:'two', 3:'three'}
```

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one',2:'two',3:'three'}  
>>> d  
  
{1:'one', 2:'two', 3:'three'}  
  
>>> print d[2]
```

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one',2:'two',3:'three'}  
>>> d  
  
{1:'one', 2:'two', 3:'three'}  
  
>>> print d[2]  
two
```

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one',2:'two',3:'three'}  
>>> d
```

```
{1:'one', 2:'two', 3:'three'}
```

```
>>> print d[2]  
two
```

Bővítés lehetséges:

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one',2:'two',3:'three'}  
>>> d
```

```
{1:'one', 2:'two', 3:'three'}
```

```
>>> print d[2]  
two
```

Bővítés lehetséges:

```
>>> d[4]='four'  
>>> d
```

dictionaries típus

Ez egy asszociációs "tömb", **kulcs** → **érték** párokkal dolgozik.

```
>>> d={1:'one',2:'two',3:'three'}  
>>> d
```

```
{1:'one', 2:'two', 3:'three'}
```

```
>>> print b[2]  
two
```

Bővítés lehetséges:

```
>>> d[4]='four'  
>>> d
```

```
{1:'one', 2:'two', 3:'three', 4:'four'}
```


Operátorok

Operátor típusok:

- aritmetikai,
- relációs,
- logikai,
- bit,
- értékadó,
- speciális.

Aritmetikai operátorok

- + összeadás operátor,
- kivonás operátor,
- * szorzás operátor,
- / osztás operátor,
- % modulo operátor (megy törtekkel is),
- // osztás alsó egészérték ($a = -7 // 2$ az -4),
- ** hatványozás operátor.

Relációs operátorok

- < kisebb operátor,
- <= kisebb egyenlő operátor,
- = egyenlő operátor,
- != nem egyenlő operátor,
- >= nagyobb egyenlő operátor,
- > nagyobb operátor.

Logikai operátorok

and logikai és operátor,
or logikai vagy operátor,
not logikai és operátor.

Bit operátorok

- & bitenkénti és operátor,
- | bitenkénti vagy operátor,
- ^ bitenkénti kizáró vagy operátor,
- ~ egyes komplementum operátor,
- >> jobbra siftelés operátor,
- << balra siftelés operátor.

Értékadó operátorok

- `=` értékadás operátor,
- `+=` összeadás rekurzív operátor,
- `-=` kivonás rekurzív operátor,
- `*=` szorzás rekurzív operátor,
- `/=` osztás rekurzív operátor,
- `%=` modulo rekurzív operátor,
- `//=` osztás alsó rekurzív egészérték,
- `**=` hatványozás rekurzív operátor.
- `&=` bitenkénti és rekurzív operátor,
- `|=` bitenkénti vagy rekurzív operátor,
- `^=` bitenkénti kizáró vagy rekurzív operátor,
- `>>=` jobbra siftelés rekurzív operátor,
- `<<=` balra siftelés rekurzív operátor.

Speciális operátorok

`is` "egyformaság" operátor,

`is not` "nem egyformaság" operátor.

Speciális operátorok

is "egyformaság" operátor,

is not "nem egyformaság" operátor.

Példa:

Speciális operátorok

is "egyformaság" operátor,

is not "nem egyformaság" operátor.

Példa:

```
>>> a=' abc'  
>>> b=a  
>>> c=' def'
```

Speciális operátorok

`is` "egyformaság" operátor,

`is not` "nem egyformaság" operátor.

Példa:

```
>>> a='abc'  
>>> b=a  
>>> c='def'  
>>> print a is b  
True
```

Speciális operátorok

`is` "egyformaság" operátor,

`is not` "nem egyformaság" operátor.

Példa:

```
>>> a='abc'
>>> b=a
>>> c='def'
>>> print a is b
True
>>> print a is not c
True
```

if utasítás

if utasítás

Klasszikus elágazás utasítás.

if utasítás

Klasszikus elágazás utasítás.

if

Az utasítás.

if utasítás

Klasszikus elágazás utasítás.

if kifejezés

Az utasítás.

Relációs kiértékelésű kifejezés.

if utasítás

Klasszikus elágazás utasítás.

if kifejezés :

Az utasítás.

Relációs kiértékelésű kifejezés.

Kettőspont.

if utasítás

Klasszikus elágazás utasítás.

```
if kifejezés :  
    → igaz ág
```

Az utasítás.

Relációs kiértékelésű kifejezés.

Kettőspont.

Az igaz ág egy tabulátorral beljebb.

if utasítás

Klasszikus elágazás utasítás.

```
if kifejezés :  
    → igaz ág
```

Az utasítás.

Relációs kiértékelésű kifejezés.

Kettőspont.

Az igaz ág egy tabulátorral beljebb.

Példa:

if utasítás

Klasszikus elágazás utasítás.

```
if kifejezés :  
    → igaz ág
```

Az utasítás.

Relációs kiértékelésű kifejezés.

Kettőspont.

Az igaz ág egy tabulátorral beljebb.

Példa:

```
>>> if 5>0 :  
...     print "Igaz"  
...  
Igaz  
>>>
```

else utasítás

else utasítás

Az úgynevezett "másik ág" utasítás.

else utasítás

Az úgynevezett "másik ág" utasítás.

```
if kifejezés :  
    → igaz ág
```

Valamilyen szerkezet meg kell,
hogy előzze az **else**-t

else utasítás

Az úgynevezett "másik ág" utasítás.

```
if kifejezés :  
    → igaz ág  
else:
```

Valamilyen szerkezet meg kell,
hogy előzze az **else**-t
Az utasítás. a : kell!

else utasítás

Az úgynevezett "másik ág" utasítás.

```
if kifejezés :  
    → igaz ág  
else:  
    → másik ág
```

Valamilyen szerkezet meg kell,
hogy előzze az **else**-t

Az utasítás. a : kell!

A másik ág, ez is **egy tabulátorral**
beljebb.

else utasítás

Az úgynevezett "másik ág" utasítás.

```
if kifejezés :  
    → igaz ág  
else:  
    → másik ág
```

Valamilyen szerkezet meg kell,
hogy előzze az **else**-t

Az utasítás. a : kell!

A másik ág, ez is **egy tabulátorral
beljebb.**

Példa:

else utasítás

Az úgynevezett "másik ág" utasítás.

```
if kifejezés :  
    → igaz ág  
else:  
    → másik ág
```

Valamilyen szerkezet meg kell,
hogy előzze az **else**-t

Az utasítás. a : kell!

A másik ág, ez is **egy tabulátorral
beljebb.**

Példa:

```
>>> if 5<0:  
...     print "Igaz"  
...else:  
...     print "Másik"  
...  
Másik
```

`elif` utasítás

`elif` utasítás

Többszörös elágazást tesz lehetővé.

`elif` utasítás

```
if kifejezés1 :  
→ ág1
```

Valamilyen szerkezet meg kell, hogy előzze az `elif`-et

`elif` utasítás

```
if kifejezés1 :  
→ ág1  
elif kifejezés2:
```

Valamilyen szerkezet meg kell, hogy előzze az **elif**-et
Az utasítás a 2. relációs kifejezéssel

elif utasítás

```
if kifejezés1 :  
→ ág1  
elif kifejezés2:  
→ ág2
```

Valamilyen szerkezet meg kell, hogy előzze az **elif**-et
Az utasítás a 2. relációs kifejezéssel
A 2. ág, ez is **egy tabulátorral beljebb**.

elif utasítás

```
if kifejezés1 :  
→ ág1  
elif kifejezés2:  
→ ág2  
  ⋮
```

Valamilyen szerkezet meg kell, hogy előzze az **elif**-et
Az utasítás a 2. relációs kifejezéssel
A 2. ág, ez is **egy tabulátorral beljebb**.
Jöhet **else**, vagy **elif**, vagy folytatódhat a program.

elif utasítás

```
if kifejezés1 :  
→ ág1  
elif kifejezés2:  
→ ág2  
  :
```

Példa:

Valamilyen szerkezet meg kell, hogy előzze az **elif**-et
Az utasítás a 2. relációs kifejezéssel
A 2. ág, ez is **egy tabulátorral beljebb**.
Jöhet **else**, vagy **elif**, vagy folytatódhat a program.

elif utasítás

```
if kifejezés1 :  
→ ág1  
elif kifejezés2:  
→ ág2  
:
```

Valamilyen szerkezet meg kell, hogy előzze az **elif**-et
Az utasítás a 2. relációs kifejezéssel
A 2. ág, ez is **egy tabulátorral beljebb**.
Jöhet **else**, vagy **elif**, vagy folytatódhat a program.

Példa:

```
>>> a=5  
>>> if a>5:  
...     print "larger"  
...elif a==5:  
...     print "equal"  
...else:  
...     print "less"  
...  
equal
```

for utasítás

Klasszikus "számláló" ciklus.

for utasítás

Klasszikus "számláló" ciklus.

for **in** **:**

Az utasítás minden "tartozékával".

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in          :
```

Az utasítás minden "tartozékával".
A "ciklusváltozó".

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in lista :
```

Az utasítás minden "tartozékával".

A "ciklusváltozó".

A lista jellegű kifejezés.

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in lista :  
→ ciklustörzs
```

Az utasítás minden "tartozékával".

A "ciklusváltozó".

A lista jellegű kifejezés.

A ciklustörzs.

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in lista :  
→ ciklustörzs
```

Példa:

```
>>> for i in [1,2,3]:  
...     print i  
...  
1  
2  
3  
>>>
```

Az utasítás minden "tartozékával".
A "ciklusváltozó".
A lista jellegű kifejezés.
A ciklustörzs.

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in lista :  
→ ciklustörzs
```

Példa:

```
>>> for i in [1,2,3]:  
...     print i  
...  
1  
2  
3  
>>>
```

Az utasítás minden "tartozékával".
A "ciklusváltozó".
A lista jellegű kifejezés.
A ciklustörzs.

A lista lehet valódi lista jellegű kifejezés, vagy:

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in lista :  
→ ciklustörzs
```

Példa:

```
>>> for i in [1,2,3]:  
...     print i  
...  
1  
2  
3  
>>>
```

Az utasítás minden "tartozékával".

A "ciklusváltozó".

A lista jellegű kifejezés.

A ciklustörzs.

A lista lehet valódi lista jellegű
kifejezés, vagy:

range függvény,

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in lista :  
→ ciklustörzs
```

Példa:

```
>>> for i in [1,2,3]:  
...     print i  
...  
1  
2  
3  
>>>
```

Az utasítás minden "tartozékával".

A "ciklusváltozó".

A lista jellegű kifejezés.

A ciklustörzs.

A lista lehet valódi lista jellegű kifejezés, vagy:

range függvény,

xrange függvény,

for utasítás

Klasszikus "számláló" ciklus.

```
for változó in lista :  
→ ciklustörzs
```

Példa:

```
>>> for i in [1,2,3]:  
...     print i  
...  
1  
2  
3  
>>>
```

Az utasítás minden "tartozékával".
A "ciklusváltozó".
A lista jellegű kifejezés.
A ciklustörzs.

A lista lehet valódi lista jellegű
kifejezés, vagy:
range függvény,
xrange függvény,
amit mi csinálunk.

range,xrange

range valós listát generál.

xrange szám generátor.

range,xrange

range valós listát generál.

range (**m**) ,

m a felső határ, ez már nem kerül a ciklusba,

xrange szám generátor.

xrange (**m**) ,

range,xrange

range valós listát generál.

```
range(m),  
range(1,m),
```

m a felső határ, ez már nem kerül a ciklusba,
1 az alsó határ, ez benne van a ciklusba,

xrange szám generátor.

```
xrange(m),  
xrange(1,m),
```

range,xrange

range valós listát generál.

```
range(m),  
range(1,m),  
range(1,m,n).
```

xrange szám generátor.

```
xrange(m),  
xrange(1,m),  
xrange(1,m,n).
```

m a felső határ, ez már nem kerül a ciklusba,
1 az alsó határ, ez benne van a ciklusba,
n a lépésnagyság.

range,xrange

range valós listát generál.

```
range(m),  
range(1,m),  
range(1,m,n).
```

xrange szám generátor.

```
xrange(m),  
xrange(1,m),  
xrange(1,m,n).
```

m a felső határ, ez már nem kerül a ciklusba,
1 az alsó határ, ez benne van a ciklusba,
n a lépésméret.

Az **m**, **1**, **n** értékek **csak egészek** lehetnek.

range, xrange példa

```
>>> for i in range(5):  
...     print i  
...  
0  
1  
2  
3  
4
```

range, xrange példa

```
>>> for i in range(5):>>> for i in range(1,5):  
...     print i           ...     print i  
...                       ...  
0                           1  
1                           2  
2                           3  
3                           4  
4
```

range, xrange példa

```
>>> for i in range(5):>>> for i in range(1,5):  
...     print i           ...     print i  
...  
...  
0               1  
1               2  
2               3  
3               4  
4
```

```
>>> for i in range(1,5,2):  
...     print i  
...  
1  
3
```

range, xrange példa

```
>>> for i in range(5): >>> for i in range(1,5): >>> for i in range(1,5,2):
...     print i          ...     print i          ...     print i
...                      ...                      ...
0                        1                        1
1                        2                        3
2                        3
3                        4
4

>>> for i in xrange(5):
...     print i
...
0
1
2
3
4
```

range, xrange példa

```
>>> for i in range(5): >>> for i in range(1,5): >>> for i in range(1,5,2):  
...     print i           ...     print i           ...     print i  
...  
0               1               1  
1               2               3  
2               3  
3               4  
4
```

```
>>> for i in xrange(5): >>> for i in xrange(1,5):  
...     print i           ...     print i  
...  
0               1  
1               2  
2               3  
3               4  
4
```

range, xrange példa

```
>>> for i in range(5): >>> for i in range(1,5): >>> for i in range(1,5,2):
...     print i           ...     print i           ...     print i
...                       ...                       ...
0                         1                         1
1                         2                         3
2                         3
3                         4
4

>>> for i in xrange(5): >>> for i in xrange(1,5): >>> for i in xrange(1,5,2):
...     print i           ...     print i           ...     print i
...                       ...                       ...
0                         1                         1
1                         2                         3
2                         3
3                         4
4
```

`while` utasítás

Előltesztelő ciklus utasítás.

while utasítás

Előltesztelő ciklus utasítás.

while : A **while** minden "tartozékával".

while utasítás

Előltesztelő ciklus utasítás.

while kifejezés :

A **while** minden "tartozékával".
Relációs jellegű kifejezés.

while utasítás

Előltesztelő ciklus utasítás.

```
while kifejezés :  
    → ciklustörzs
```

A **while** minden "tartozékával".
Relációs jellegű kifejezés.
A ciklustörzs.

while utasítás

Előltesztelő ciklus utasítás.

```
while kifejezés :  
    → ciklustörzs
```

A **while** minden "tartozékával".
Relációs jellegű kifejezés.
A ciklustörzs.

Példa:

while utasítás

Előltesztelő ciklus utasítás.

```
while kifejezés :  
    → ciklustörzs
```

A **while** minden "tartozékával".
Relációs jellegű kifejezés.
A ciklustörzs.

Példa:

```
>>> a = 0  
>>> while a < 3 :  
...     print i  
...     a = a + 1  
...  
0  
1  
2
```

break utasítás

Az aktuális ciklusból kilép.

break utasítás

Az aktuális ciklusból kilép.

Példa1:

```
>>> for i in [0,1,2,3,4]:  
...     if i == 2 :  
...         break  
...     print i  
...  
0  
1
```

break utasítás

Az aktuális ciklusból kilép.

Példa1:

```
>>> for i in [0,1,2,3,4]:  
...     if i == 2 :  
...         break  
...     print i  
...  
0  
1
```

Példa2:

```
>>> for i in [1,2,3] :  
...     for j in ['a','b','c'] :  
...         if j == 'b' :  
...             break  
...         print i,j  
...  
1 a  
2 a  
3 a
```


continue utasítás

Az aktuális ciklusban az utasítás hatására a program a **ciklustörzs hátralevő részét kihagyja**.

continue utasítás

Az aktuális ciklusban az utasítás hatására a program a **ciklustörzs hátralevő részét kihagyja**.

Példa:

```
>>> for i in range(5) :  
...     if i == 3 :  
...         continue  
...     print i  
...  
0  
1  
2  
4
```

pass utasítás

Üres utasítás. Akkor használatos, ha **szintaktikai okokból** kell valamijen kifejezés.

pass utasítás

Üres utasítás. Akkor használatos, ha **szintaktikai okokból** kell valamijen kifejezés.

```
>>> for i in range(10) :  
...     pass  
...
```

pass utasítás

Üres utasítás. Akkor használatos, ha **szintaktikai okokból** kell valamijen kifejezés.

```
>>> for i in range(10) :  
...     pass  
...
```

Ez bármilyen logikai blokkra használható, amely lehet:

pass utasítás

Üres utasítás. Akkor használatos, ha **szintaktikai okokból** kell valamijen kifejezés.

```
>>> for i in range(10) :  
...     pass  
...
```

Ez bármilyen logikai blokkra használható, amely lehet:

- igaz ág,
- másik ág,
- ciklus törzs,
- függvény törzs,
- és szekvencia.

else utasítás a ciklusokban

Bármilyen ciklus végére tehető **else** utasítás.

else utasítás a ciklusokban

Bármilyen ciklus végére tehető **else** utasítás.

Példa1:

```
>>> a=5
>>> for i in range(5):
...     if i==a:
...         break
...else:
...     print "Nincs benne"
...
Nincs benne
>>>
```


else utasítás a ciklusokban

Bármilyen ciklus végére tehető **else** utasítás.

Példa1:

```
>>> a=5
>>> for i in range(5):
...     if i==a:
...         break
...else:
...     print "Nincs benne"
...
Nincs benne
>>>
```

Példa2:

```
>>> a=4
>>> for i in range(5):
...     if i==a:
...         break
...else:
...     print "Nincs benne"
...
>>>
```

else utasítás a ciklusokban

Bármilyen ciklus végére tehető **else** utasítás.

Példa1:

```
>>> a=5
>>> for i in range(5):
...     if i==a:
...         break
...else:
...     print "Nincs benne"
...
Nincs benne
>>>
```

Példa2:

```
>>> a=4
>>> for i in range(5):
...     if i==a:
...         break
...else:
...     print "Nincs benne"
...
>>>
```

A **break** végrehajtásakor az **else** utasítást is kikerüli a program.

Függvények

Megjegyzés

Eddig használhattuk a python shell-t, (>>>) de a függvényeknél célszerű már programot írni.

Függvények

Megjegyzés

Eddig használhattuk a python shell-t, (>>>) de a függvényeknél célszerű már programot írni.

Tehát vagy:

```
python program.py
```

Függvények

Megjegyzés

Eddig használhattuk a python shell-t, (>>>) de a függvényeknél célszerű már programot írni.

Tehát vagy:

```
python program.py
```

Vagy a program első sora:

```
#!/usr/bin/python
```

Függvények

A **def** és a **()** : kötelező,

```
def függvénynév (           ) :
```

Függvények

`def` függvénynév () :

A `def` és a `()` : kötelező,
a függvény neve, ez is
kötelező, de ezt mi adjuk,

Függvények

```
def függvénynév ( paraméterek ) :
```

A **def** és a **()** : kötelező,
a függvény neve, ez is
kötelező, de ezt mi adjuk,
paraméter lista, ha kell,

Függvények

```
def függvénynév ( paraméterek ) :  
    → """doc string"""
```

A **def** és a **()** : kötelező,
a függvény neve, ez is
kötelező, de ezt mi adjuk,
paraméter lista, ha kell,
dokumentációs sztring, ha úgy
gondoljuk,

Függvények

```
def függvénynév ( paraméterek ) :  
    → """doc string"""  
    → változók
```

A **def** és a **()** : kötelező,
a függvény neve, ez is
kötelező, de ezt mi adjuk,
paraméter lista, ha kell,
dokumentációs sztring, ha úgy
gondoljuk,
a függvény lokális változói, ha
kellenek,

Függvények

```
def függvéynév ( paraméterek ) :  
    → """doc string"""  
    → változók  
    → függvénytörzs  
    → :  
    → :  
    → :  
    → :
```

A **def** és a **()** : kötelező,
a függvény neve, ez is
kötelező, de ezt mi adjuk,
paraméter lista, ha kell,
dokumentációs sztring, ha úgy
gondoljuk,
a függvény lokális változói, ha
kellenek,
a függvény törzse,

Függvények

```
def függvénynév ( paraméterek ) :  
    → """doc string"""  
    → változók  
    → függvénytörzs  
    → :  
    → :  
    → :  
    → :  
    → return par
```

A **def** és a **()** : kötelező,
a függvény neve, ez is
kötelező, de ezt mi adjuk,
paraméter lista, ha kell,
dokumentációs sztring, ha úgy
gondoljuk,
a függvény lokális változói, ha
kellenek,
a függvény törzse,
return utasítás paraméterrel,
ha kell.

Változók 1.

Belső és külső változók kezelése, példa:

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    x="inside"  
    print x
```

Képernyő



```
x="outside"
```

```
fgv()  
print x
```

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    x="inside"  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    x="inside"  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.
Külső változó.

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    x="inside"  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.

Külső változó.

Függvény hívás.

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv() :  
    x="inside"  
    print x
```

Képernyő



```
x="outside"
```

```
fgv()  
print x
```

A futtató program neve.

Külső változó.

Függvény hívás.

A függvény fejléce.

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv() :  
    x="inside"  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.

Külső változó.

Függvény hívás.

A függvény fejléce.

Belső változó.

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv() :  
    x="inside"  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.

Külső változó.

Függvény hívás.

A függvény fejléce.

Belső változó.

Printelés belül.

Változók 1.

Belső és külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv() :  
    x="inside"  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő

```
inside  
outside
```

A futtató program neve.

Külső változó.

Függvény hívás.

A függvény fejléce.

Belső változó.

Printelés belül.

Printelés kívül.

Változók 2.

Külső változók kezelése, példa:

Változók 2.

Külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



Változók 2.

Külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.

Változók 2.

Külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.
Külső változó.

Változók 2.

Külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.

Külső változó.

Függvény hívás.

Változók 2.

Külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv() :  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő



A futtató program neve.

Külső változó.

Függvény hívás.

A függvény fejléce.

Változók 2.

Külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő
outside

A futtató program neve.

Külső változó.

Függvény hívás.

A függvény fejléce.

Printelés belül.

Változók 2.

Külső változók kezelése, példa:

```
#!/usr/bin/python
```

```
def fgv():  
    print x
```

```
x="outside"
```

```
fgv()  
print x
```

Képernyő

```
outside  
outside
```

A futtató program neve.

Külső változó.

Függvény hívás.

A függvény fejléce.

Printelés belül.

Printelés kívül.

Példa: rekurzív függvény

Példa: rekurzív függvény

```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```

Példa: rekurzív függvény

Befelé ⇒

```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```

fő

Példa: rekurzív függvény

Befelé ⇒

```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```

fő
fact(4)

Példa: rekurzív függvény

Befelé ⇒

```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```

fő
fact(4)
r=1

Példa: rekurzív függvény

Befelé ⇒

```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```

fő
fact(4)
r=1

Példa: rekurzív függvény

Befelé ⇒

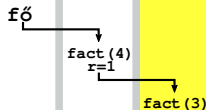
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```

fő
fact(4)
r=1

Példa: rekurzív függvény

Befelé ⇒

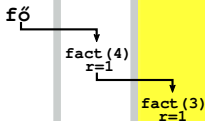
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

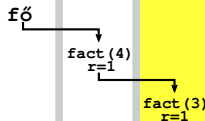
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

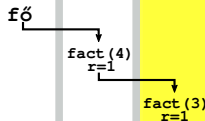
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

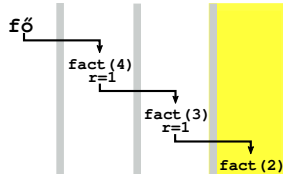
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

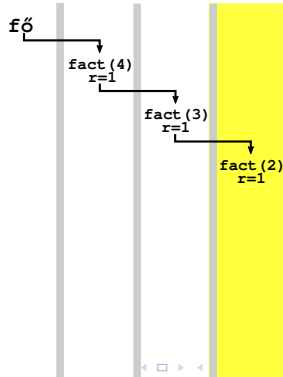
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

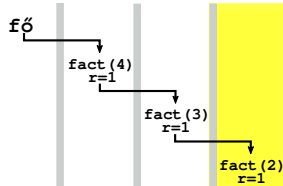
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

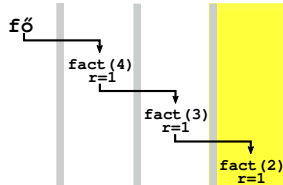
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

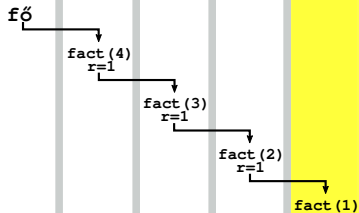
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

Befelé ⇒

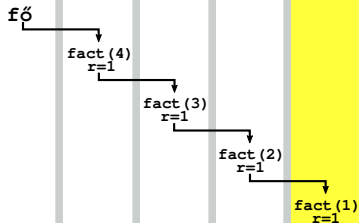
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

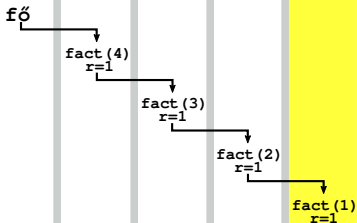
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

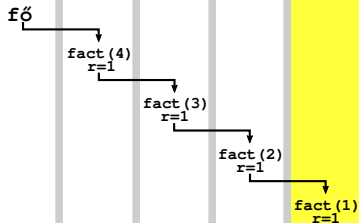
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

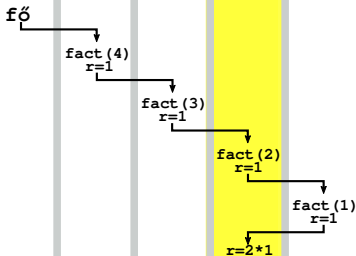
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

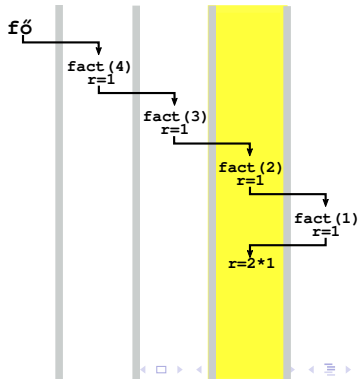
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

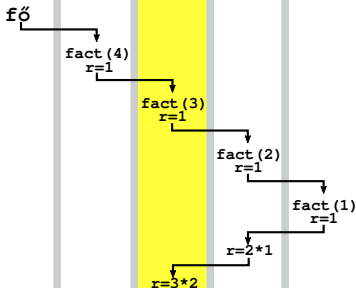
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

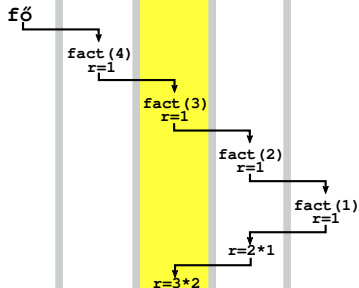
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

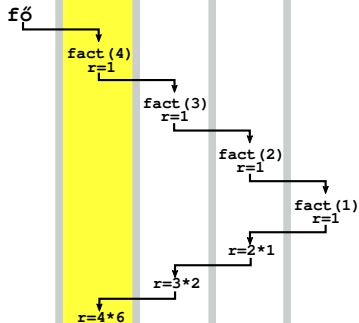
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

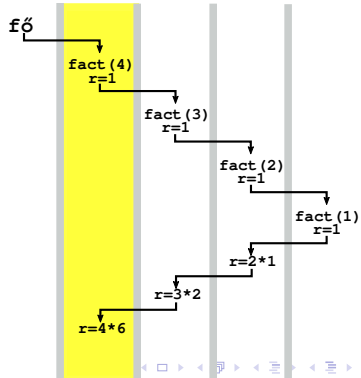
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

← Kifelé

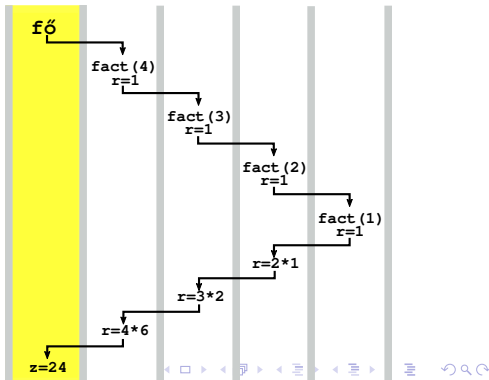
```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

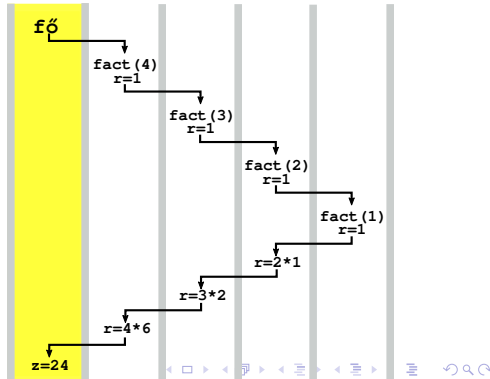
← Kifelé

```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Példa: rekurzív függvény

```
#!/usr/bin/python
def fact(n):
    r=1
    if(n>1):
        r=n*fact(n-1)
    return r
z=fact(4)
print z
```



Modulok

A modul egy olyan szöveges file, ami Python definíciókat és utasításokat tartalmaz.

Modulok

A modul egy olyan szöveges file, ami Python definíciókat és utasításokat tartalmaz.

A modul neve egyben annak a fájlnek neve, amiben létrehozzuk. A kiterjesztése: .py.

Modulok

A modul egy olyan szöveges file, ami Python definíciókat és utasításokat tartalmaz.

A modul neve egyben annak a fájlnek neve, amiben létrehozzuk. A kiterjesztése: .py.

Az aktuális modul neve a programból a

`__name__`

változóból érhető el, mint karakterlánc.

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A `fgv.py` modul.

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv1():
    print "fgv1"
```

A `fgv.py` modul.
Két függvényt tartalmaz:

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv1():
    print "fgv1"
```

A `fgv.py` modul.
Két függvényt tartalmaz:
`fgv1()` -t és

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A `fgv.py` modul.
Két függvényt tartalmaz:
 `fgv1()` -t és
 `fgv2()` -t.

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv1():
    print "fgv1"
```

A `fgv.py` modul.
Két függvényt tartalmaz:
`fgv1()` -t és
`fgv2()` -t.

Írjunk egy programot és használjuk!

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A `fgv.py` modul.
Két függvényt tartalmaz:
`fgv1()` -t és
`fgv2()` -t.

Írjunk egy programot és használjuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

A futtató program.

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A `fgv.py` modul.
Két függvényt tartalmaz:
`fgv1()` -t és
`fgv2()` -t.

Írjunk egy programot és használjuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

A futtató program.
A modul beolvasása.

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A `fgv.py` modul.
Két függvényt tartalmaz:
`fgv1()` -t és
`fgv2()` -t.

Írjunk egy programot és használjuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

A futtató program.
A modul beolvasása.
Az első függvény hívása a modulból.

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A `fgv.py` modul.
Két függvényt tartalmaz:
`fgv1()` -t és
`fgv2()` -t.

Írjunk egy programot és használjuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

A futtató program.
A modul beolvasása.
Az első függvény hívása a modulból.
A második függvény hívása a modulból.

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

Futtassuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

Futtassuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

Képernyő



Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

Futtassuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

Képernyő



Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv1():
    print "fgv1"
```

Futtassuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

Képernyő



Példa:

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

Futtassuk!

```
#!/usr/bin/python
import fgv
fgv.fgv1()
fgv.fgv2()
```

Képernyő

```
fgv1
fgv2
```

Sorok függvényen kívül

Egy modulon belül lehetnek kifejezések, amelyek **nincsenek** függvényben.

Sorok függvényen kívül

Egy modulon belül lehetnek kifejezések, amelyek **nincsenek** függvényben.

Ezek csak a program indulásánál kerülnek végrehajtásra.

Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

Az `out.py` modul.

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

Az `out.py` modul.

Kifejezés minden függvényen kívül.

Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

Az `out.py` modul.

Kifejezés minden függvényen kívül.

A modul függvénye

Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

```
#!/usr/bin/python
import out
for i in range(3):
    out.fgv()
```

Az `out.py` modul.

Kifejezés minden függvényen kívül.

A modul függvénye

A futtató program.

Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

```
#!/usr/bin/python
import out
for i in range(3):
    out.fgv()
```

Az `out.py` modul.

Kifejezés minden függvényen kívül.

A modul függvénye

A futtató program.

A modul behívása.

Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

```
#!/usr/bin/python
import out
for i in range(3):
    out.fgv()
```

Az `out.py` modul.

Kifejezés minden függvényen kívül.

A modul függvénye

A futtató program.

A modul behívása.

A modul függvény hívása ciklikusan.

Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

```
#!/usr/bin/python
import out
for i in range(3):
    out.fgv()
```

Futtassuk!

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Futtassuk!

Képernyő



Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Futtassuk!

Képernyő



Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

```
#!/usr/bin/python
import out
for i in range(3):
    out.fgv()
```

Futtassuk!

Képernyő



Sorok függvényen kívül

```
# out.py
print "Outside"
def fgv():
    print "Inside"
```

```
#!/usr/bin/python
import out
for i in range(3):
    out.fgv()
```

Futtassuk!

Képernyő

Outside

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Futtassuk!

Képernyő

Outside

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Futtassuk!

Képernyő

Outside
Inside

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Futtassuk!

Képernyő

Outside
Inside

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Futtassuk!

Képernyő

Outside
Inside
Inside

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Futtassuk!

Képernyő

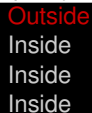
```
Outside  
Inside  
Inside
```

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Képernyő



```
Outside  
Inside  
Inside  
Inside
```

Sorok függvényen kívül

```
# out.py  
print "Outside"  
def fgv():  
    print "Inside"
```

```
#!/usr/bin/python  
import out  
for i in range(3):  
    out.fgv()
```

Képernyő

```
Outside  
Inside  
Inside  
Inside
```

Tehát a modulban található függvényen kívüli kifejezések **csak a modul betöltésénél** és **csak egyszer** kerülnek végrehajtásra.

Változók függvényen kívül

Egy modulban lehetnek olyan változók, amelyek minden függvényen kívül találhatók.

Változók függvényen kívül

Egy modulban lehetnek olyan változók, amelyek minden függvényen kívül találhatók.

Ezek a változók globális változóként szerepelnek.

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

Az `noin.py` modul.

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

Az **noin.py** modul.

Változó minden függvényen kívül.

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

Az **noin.py** modul.

Változó minden függvényen kívül.

A modul függvénye

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Az **noin.py** modul.

Változó minden függvényen kívül.

A modul függvénye

A futtató program.

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

Az **noin.py** modul.

Változó minden függvényen kívül.

A modul függvénye

```
#!/usr/bin/python  
import noin  
print noin.v  
noin.fgv()
```

A futtató program.

A modul behívása.

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Az **noin.py** modul.

Változó minden függvényen kívül.

A modul függvénye

A futtató program.

A modul behívása.

A modul külső változó printelése.

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

A futtató program.

A modul behívása.

A modul külső változó printelése.

A modul függvény hívása.

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Futtassuk!

Képernyő



Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import noin  
print noin.v  
noin.fgv()
```

Futtassuk!

Képernyő



Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Futtassuk!

Képernyő



Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Futtassuk!

Képernyő
Outside

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Futtassuk!

Képernyő

Outside
Outside

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Képernyő

Outside
Outside

Változók függvényen kívül

```
# noin.py  
v="Outside"  
def fgv():  
    print v
```

```
#!/usr/bin/python  
import out  
print noin.v  
noin.fgv()
```

Képernyő

Outside
Outside

A kérdéses változó úgy viselkedik, mint globális változó.

Modul szimbólumtábla átmozgatás

Lehetőség van adott függvények és változók nevének importálása a programba.

Példa:

Modul szimbólumtábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

Modul szimbólumtábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

```
#!/usr/bin/python
from fgv import fgv1 fgv2
fgv1()
fgv2()
```

A futtató program.

Modul szimbólumtábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

```
#!/usr/bin/python
from fgv import fgv1 fgv2
fgv1()
fgv2()
```

A futtató program.
Az importáló sor.

Modul szimbólumtábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

```
#!/usr/bin/python
from fgv import fgv1 fgv2
fgv1()
fgv2()
```

A modul, aminek a szimbólum tábláját importáljuk.

A futtató program.
Az importáló sor.
A függvények hívása.

Modul teljes szimbólum tábla átmozgatás

Lehetőség van adott függvények és változók nevének importálása a programba.

Példa:

Modul teljes szimbólum tábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

Modul teljes szimbólum tábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

```
#!/usr/bin/python
from fgv import *
fgv1()
fgv2()
```

A futtató program.

Modul teljes szimbólum tábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

```
#!/usr/bin/python
from fgv import *
fgv1()
fgv2()
```

A futtató program.
Az importáló sor.

Modul teljes szimbólum tábla átmozgatás

```
# fgv.py
def fgv1():
    print "fgv1"
def fgv2():
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

```
#!/usr/bin/python
from fgv import *
fgv1()
fgv2()
```

A futtató program.

Az importáló sor.

A függvények hívása.

Modul teljes szimbólum tábla átmozgatás

```
# fgv.py  
def fgv1():  
    print "fgv1"  
def fgv2():  
    print "fgv2"
```

A modul, aminek a szimbólum tábláját importáljuk.

```
#!/usr/bin/python  
from fgv import *  
fgv1()  
fgv2()
```

A futtató program.
Az importáló sor.
A függvények hívása.

Ez a mód minden szimbólumot átmásol **kivéve a __** karakterekkel kezdődőöket. (2db)

Csomagok

A csomagok lehetőséget nyújtanak, a modulok és ezzel a névterek struktúrálására.

Csomagok

A csomagok lehetőséget nyújtanak, a modulok és ezzel a névterek struktúrálására.

Példa:

Szeretnénk egy olyan modulrendszert összeállítani, amely logikailag összetartozó feladatokat lát el, de funkcionálisan különbözik.

Csomagok

A csomagok lehetőséget nyújtanak, a modulok és ezzel a névterek struktúrálására.

Példa:

Szeretnénk egy olyan modulrendszert összeállítani, amely logikailag összetartozó feladatokat lát el, de funkcionálisan különbözik.

Legyen ez különböző geometriai alakzatok tulajdonságainak kiszámítása.

Csomagok

A csomagok lehetőséget nyújtanak, a modulok és ezzel a névterek struktúrálására.

Példa:

Szeretnénk egy olyan modulrendszert összeállítani, amely logikailag összetartozó feladatokat lát el, de funkcionálisan különbözik.

Legyen ez különböző geometriai alakzatok tulajdonságainak kiszámítása.

Valahogy így:

Csomagok

A csomagok lehetőséget nyújtanak, a modulok és ezzel a névterek struktúrálására.

Példa:

Szeretnénk egy olyan modulrendszert összeállítani, amely logikailag összetartozó feladatokat lát el, de funkcionálisan különbözik.

Legyen ez különböző geometriai alakzatok tulajdonságainak kiszámítása.

Valahogy így:

```
geometry.rectangle.perimeter
```

Csomagok

A csomagok lehetőséget nyújtanak, a modulok és ezzel a névterek struktúrálására.

Példa:

Szeretnénk egy olyan modulrendszert összeállítani, amely logikailag összetartozó feladatokat lát el, de funkcionálisan különbözik.

Legyen ez különböző geometriai alakzatok tulajdonságainak kiszámítása.

Valahogy így:

```
geometry.rectangle.perimeter
```

```
geometry.circle.area
```

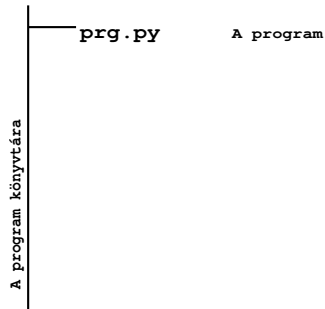
Csomagok

Az előző példához hasonló struktúrát egy katalógus rendszerben tárolja a Python.

Csomagok

Az előző példához hasonló struktúrát egy katalógus rendszerben tárolja a Python.

A program.

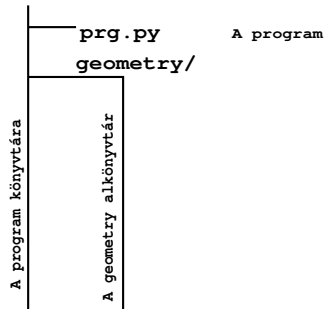


Csomagok

Az előző példához hasonló struktúrát egy katalógus rendszerben tárolja a Python.

A program.

Az "al"modulok könyvtára.



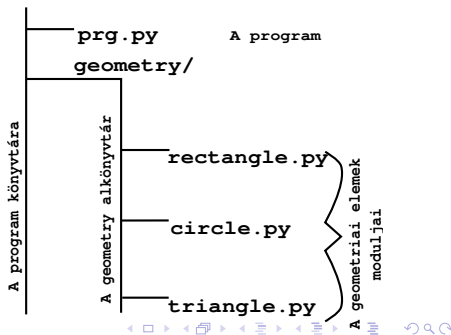
Csomagok

Az előző példához hasonló struktúrát egy katalógus rendszerben tárolja a Python.

A program.

Az "al"modulok könyvtára.

A geometriai modulok.



Csomagok

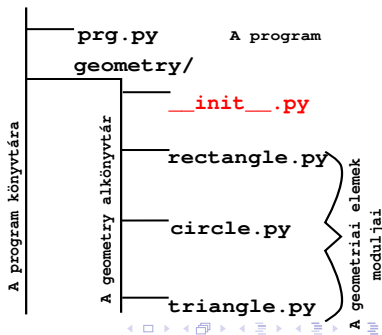
Az előző példához hasonló struktúrát egy katalógus rendszerben tárolja a Python.

A program.

Az "al"modulok könyvtára.

A geometriai modulok.

`__init__.py`



Csomagok

Az `__init__.py` fájl minden olyan könyvtárban kötelező, ahol a csomagként kezelt modulok vannak.

Csomagok

Az `__init__.py` fájl minden olyan könyvtárban kötelező, ahol a csomagként kezelt modulok vannak.

Az `__init__.py` lehet üres is, de tartalmazhat kódot.

Csomagok

Az `__init__.py` fájl minden olyan könyvtárban kötelező, ahol a csomagként kezelt modulok vannak.

Az `__init__.py` lehet üres is, de tartalmazhat kódot.

Abban a könyvtárban, ahol a csomagként kezelt modulok vannak, az összes `.py` végződésű, forrást tartalmazó fájlt binárisá fordítja a rendszer.

Csomagok

Az `__init__.py` fájl minden olyan könyvtárban kötelező, ahol a csomagként kezelt modulok vannak.

Az `__init__.py` lehet üres is, de tartalmazhat kódot.

Abban a könyvtárban, ahol a csomagként kezelt modulok vannak, az összes `.py` végződésű, forrást tartalmazó fájlt binárisá fordítja a rendszer.

Ezek végződése `.pyc` lesz.

Csomagok

Az `__init__.py` fájl minden olyan könyvtárban kötelező, ahol a csomagként kezelt modulok vannak.

A `geometry` könyvtár az első futás után:

Az `__init__.py` lehet üres is, de tartalmazhat kódot.

Abban a könyvtárban, ahol a csomagként kezelt modulok vannak, az összes `.py` végződésű, forrást tartalmazó fájlt binárisá fordítja a rendszer.

Ezek végződése `.pyc` lesz.

Csomagok

Az `__init__.py` fájl minden olyan könyvtárban kötelező, ahol a csomagként kezelt modulok vannak.

Az `__init__.py` lehet üres is, de tartalmazhat kódot.

Abban a könyvtárban, ahol a csomagként kezelt modulok vannak, az összes `.py` végződésű, forrást tartalmazó fájl binárisá fordítja a rendszer.

Ezek végződése `.pyc` lesz.

A `geometry` könyvtár az első futás után:

```
__init__.py  
__init__.pyc  
circle.py  
circle.pyc  
rectangle.py  
rectangle.pyc  
triangle.py  
triangle.pyc
```

A csomag részeinek importálása

A csomag moduljai egyenként is importálhatók:

A csomag részeinek importálása

A csomag moduljai egyenként is importálhatók:

```
import geometry.circle
```

A csomag részeinek importálása

A csomag moduljai egyenként is importálhatók:

```
import geometry.circle
```

Ekkor a hivatkozást teljes útvonallal kell megadni.

A csomag részeinek importálása

A csomag moduljai egyenként is importálhatók:

```
import geometry.circle
```

Ekkor a hivatkozást teljes útvonallal kell megadni.

```
from geometry import circle
```

A csomag részeinek importálása

A csomag moduljai egyenként is importálhatók:

```
import geometry.circle
```

Ekkor a hivatkozást teljes útvonallal kell megadni.

```
from geometry import circle
```

Ekkor nem kell a **geometry** előtag.

Bevezető
Változók
Operátorok
Utasítások
Függvények
Modulok

Modul
Kifejezések függvényen kívül
Modul szimbólumtábla átmozgatás
Csomagok

”Álnevek” használata

”Álnevek” használata

Néha a modul neve hosszú, vagy az elérési útja nagyon bonyolult.

”Álnevek” használata

Néha a modul neve hosszú, vagy az elérési útja nagyon bonyolult.

Ilyenkor lehetőség van ”álnév” használatára.

Példa:

”Álnevek” használata

Néha a modul neve hosszú, vagy az elérési útja nagyon bonyolult.

Ilyenkor lehetőség van ”álnév” használatára.

Példa:

```
import geometry.circle
```

```
import geometry.circle as circle
```

”Álnevek” használata

Néha a modul neve hosszú, vagy az elérési útja nagyon bonyolult.

Ilyenkor lehetőség van ”álnév” használatára.

Példa:

```
import geometry.circle
```

```
import geometry.circle as circle
```

Ekkor a terület függvény elérése:

”Álnevek” használata

Néha a modul neve hosszú, vagy az elérési útja nagyon bonyolult.

Ilyenkor lehetőség van ”álnév” használatára.

Példa:

```
import geometry.circle
```

```
import geometry.circle as circle
```

Ekkor a kerület függvény elérése:

```
geometry.circle.perimeter()
```

```
circle.perimeter()
```

”Álnevek” használata

Néha a modul neve hosszú, vagy az elérési útja nagyon bonyolult.

Ilyenkor lehetőség van ”álnév” használatára.

Példa:

```
import geometry.circle
```

```
import geometry.circle as circle
```

Ekkor a kerület függvény elérése:

```
geometry.circle.perimeter()
```

```
circle.perimeter()
```