



# Introduction to Programming

## Lesson 3

# Outline

- Importing Modules
- Abstraction
- User defined functions
- Parameter passing
- Assignment and mutability

# Lesson Code

[goo.gl/1uCTgV](https://goo.gl/1uCTgV)

# importing modules

```
>>> dir(__builtins__)
```

- Built-in functions and classes

(“int”, “len”, “sum”, “range”, “min”, “max”)

- Core python functions

# Python Standard Library(PSL)

# Python Standard Library(PSL)

Ավելի շատ function-ներ և class-եր կան PSL-ում

# Python Standard Library(PSL)

Ավելի շատ function-ներ և class-եր կան PSL-ում

- Network programming
- Database programming
- Mathematical functions
- Pseudorandom generator
- ...

# Python Standard Library(PSL)

Ավելի շատ function-ներ և class-եր կան PSL-ում

- Network programming
- Database programming
- Mathematical functions
- Pseudorandom generator
- ...

The PSL-ի function-ները և class-երը դասավորված են module/package-ների մեջ



# PSL module math

`sqrt()` սահմանված է PSL-ի `math`  
`module-ում`

# PSL module math

sqrt() սահմանված է PSL-ի math  
module-ում

Module իմպորտ անելու համար:

```
import <module>
```

```
>>> import math  
>>>
```

# PSL module math

`sqrt()` սահմանված է PSL-ի `math` `module`-ում

Module իմպորտ անելու համար:

```
import <module>
```

`math.sqrt()`

```
>>> import math
>>> math.sqrt(4)
2.0
>>> sqrt(4)
Traceback (most recent call
last):
  File "<pyshell#10>", line
1, in <module>
    sqrt(4)
NameError: name 'sqrt' is
not defined
>>>
```

# PSL module math

`sqrt()` սահմանված է PSL-ի `math` module-ում

Module իմպորտ անելու համար:

```
import <module>
```

`math.sqrt()`

`math` module-ը պարունակում է մաթ ֆունկցիաներ և հաստատուններ

```
>>> import math
>>> math.sqrt(4)
2.0
>>> sqrt(4)
Traceback (most recent call
last):
  File "<pyshell#10>", line
  1, in <module>
    sqrt(4)
NameError: name 'sqrt' is
not defined
>>> help(math)
Help on module math:

...
>>> math.cos(0)
1.0
>>> math.log(8)
2.0794415416798357
>>> math.log(8, 2)
3.0
>>> math.pi
3.141592653589793
```

# PSL module math

# PSL module math

```
>>> import math
```

```
>>> math.pi
```

```
3.1415926535897931
```

```
>>> math.cos(0)
```

```
1.0
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

# PSL module math

```
>>> import math
```

```
>>> math.pi
```

```
3.1415926535897931
```

```
>>> math.cos(0)
```

```
1.0
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

```
>>> dir(math)
```

```
['__doc__', '__file__', '__name__', '__package__',  
..., 'cos',  
'cosh', 'degrees', 'e', 'exp', ..., 'pi', 'pow',  
  'radians', 'sin', 'sinh', 'sqrt', 'tan',  
'tanh', 'trunc']
```

# PSL module math

```
>>> import math
```

```
>>> math.pi
```

```
3.1415926535897931
```

```
>>> math.cos(0)
```

```
1.0
```

```
>>> math.cos(math.pi)
```

```
-1.0
```

```
>>> dir(math)
```

```
['__doc__', '__file__', '__name__', '__package__',  
..., 'cos',  
'cosh', 'degrees', 'e', 'exp', ..., 'pi', 'pow',  
  'radians', 'sin', 'sinh', 'sqrt', 'tan',  
'tanh', 'trunc']
```

```
>>> help(math)
```

```
>>> help(math.cos)
```



# Exercise 1

Գրեք ծրագիր որը հաշվում է

- a) Ուղղանկյուն եռանկյան ներքնաձիգի երկարությունը  
երբ  $a=3$ ,  $b=4$
- b) Շրջանի մակերեսը  
երբ  $r=10$

# Exercise 1

Գրեք ծրագիր որը հաշվում է

- a) Ուղղանկյուն եռանկյան ներքնաձիգի երկարությունը  
երբ  $a=3$ ,  $b=4$
- b) Շրջանի մակերեսը  
երբ  $r=10$

```
import math
```

```
a = 3
```

```
b = 4
```

```
r = 10
```

```
c = math.sqrt(a**2 + b**2)
```

```
s = math.pi*r**2
```

ex31.py

# importing modules

Module import անելու տարբեր ձևեր կան:

# importing modules

Module import անելու տարբեր ձևեր կան:

```
import somemodule
```

# importing modules

Module import անելու տարբեր ձևեր կան:

```
import somemodule  
from somemodule import *
```

# importing modules

Module import անելու տարբեր ձևեր կան:

```
import somemodule  
from somemodule import *  
from somemodule import name
```

# importing modules

Module import անելու տարբեր ձևեր կան:

```
import somemodule  
from somemodule import *  
from somemodule import name  
from somefile import name as nm
```

# importing modules



# importing modules

```
>>> dir()  
['__builtins__', '__doc__', '__loader__', '__name__',  
  '__package__', '__spec__', 'cls']
```

# importing modules

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls']
>>> import math
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls', 'math']
```

# importing modules

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls']
>>> import math
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls', 'math']
>>> from math import cos
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls', 'cos', 'math']
>>> cos(0)
1.0
```

# importing modules

```
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls']
>>> import math
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls', 'math']
>>> from math import cos
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls', 'cos', 'math']
>>> cos(0)
1.0
>>> from math import sin, pi
>>> dir()
['__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'cls', 'cos', 'math', 'pi', 'sin']
>>> sin(pi)
1.2246467991473532e-16
```

# time module

```
import time
```

```
time.daylight
```

```
0
```

```
time.gmtime()
```

```
time.struct_time(tm_year=2014, tm_mon=12, tm_mday=5, tm_hour=10, tm_min=2, tm_sec=41, tm_wday=4, tm_yday=339, tm_isdst=0)
```

```
time.localtime()
```

```
time.struct_time(tm_year=2014, tm_mon=12, tm_mday=5, tm_hour=14, tm_min=3, tm_sec=48, tm_wday=4, tm_yday=339, tm_isdst=0)
```

```
time.sleep(5) # հիմա վայրկյան ոչինչ մի արա
```

# sys module

```
import sys
```

```
sys.platform
```

```
sys.version
```

```
sys.version_info
```

```
print("Start")  
sys.exit()  
print("Never Printed!")
```

```
print(dir(sys))
```

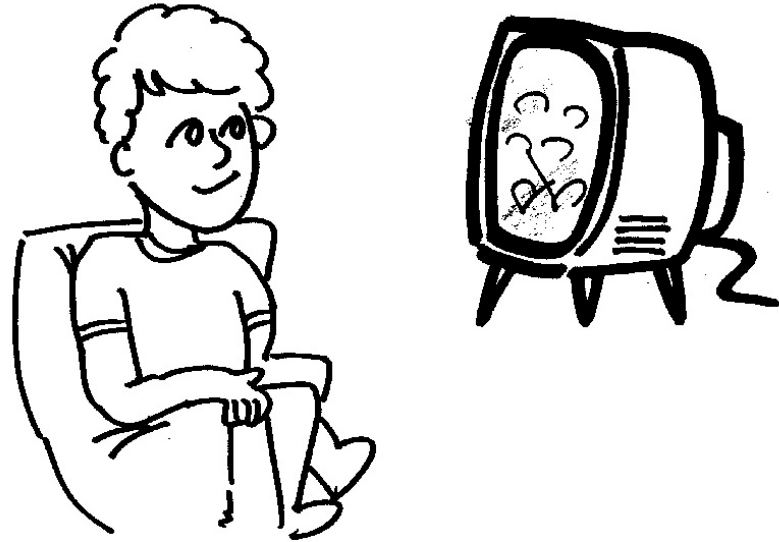
```
sys.path
```

# Ինչ է նշականում Abstraction?

- Informal description
  - “Ինձ մի անհանգստացրեք մանրունքներով”
  - Դուք փորձում եք համառոտ, ամփոփ (աբստրակտ) նկարագրություն ստանալ

# Informal explanation of abstraction

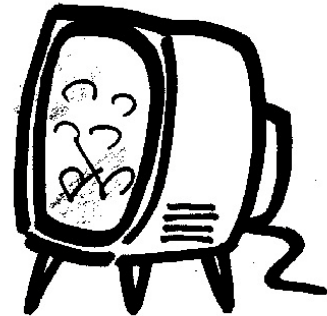
- Երբ գնում եք TV





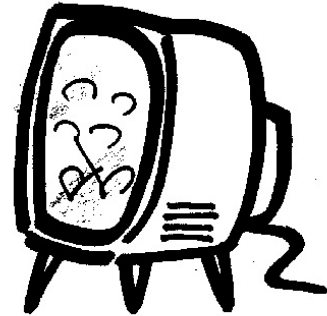
# Informal explanation of abstraction

- Երբ գնում եք TV
- Ձեզ անհրաժեշտ է իմանալ
  - միացնել
  - անջատել
  - ալիք փոխել

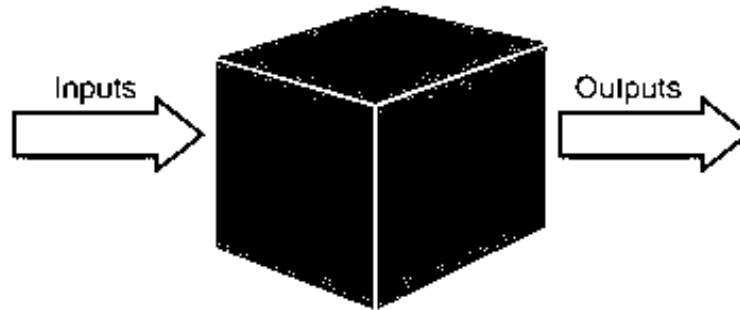


# Informal explanation of abstraction

- Երբ գնում եք TV
- Ձեզ անհրաժեշտ է իմանալ
  - միացնել
  - անջատել
  - ալիք փոխել
- Ձեզ **պետք չէ** իմանալ թե ինչպես է այն աշխատում

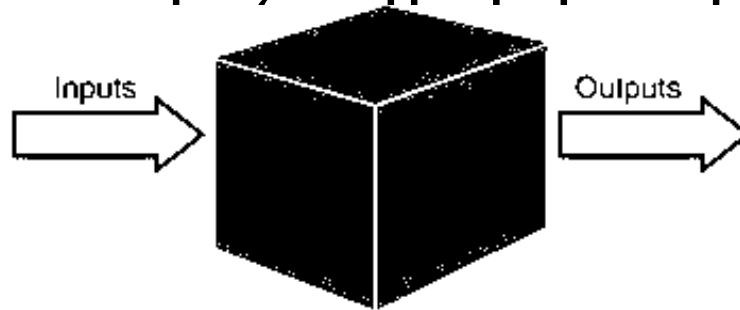


# Procedural abstraction



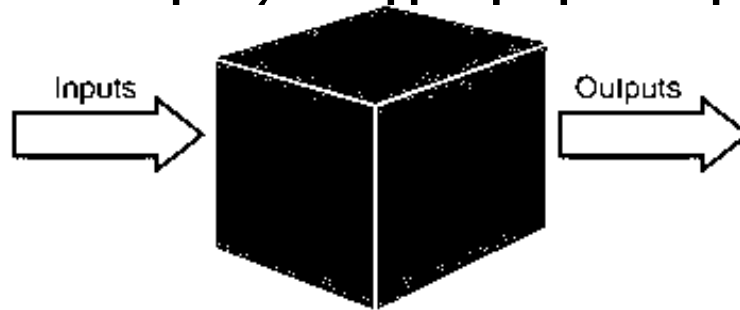
# Procedural abstraction

- Give a name to a piece of code that does something (անուն տուր code-ին, որը կատարում է որոշակի գործողություն/ներ)



# Procedural abstraction

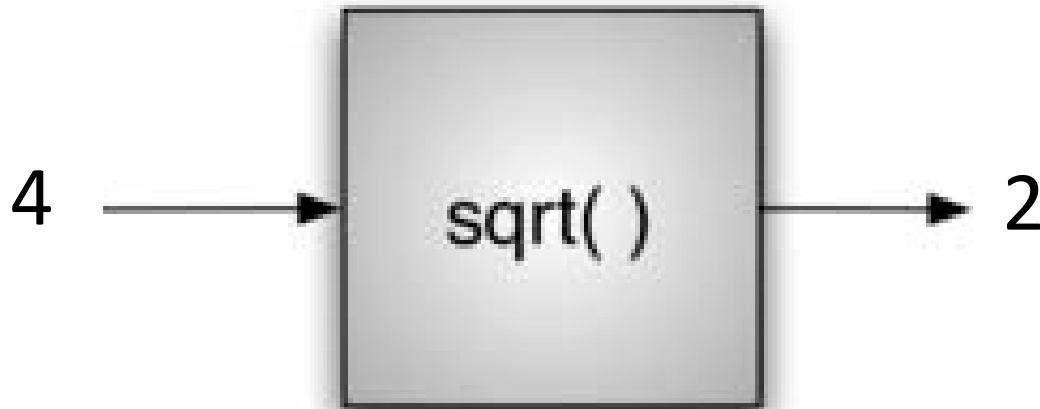
- Give a name to a piece of code that does something (անուն տուր code-ին, որը կատարում է որոշակի գործողություն/ներ)



- use the name to get that something done (օգտագործիր այդ անունը անելու այդ գործողություն/ները)

# Procedural abstraction

```
>>> from math import sqrt  
>>> sqrt(4)  
>>> 2
```



# Defining functions

# Defining functions

define function

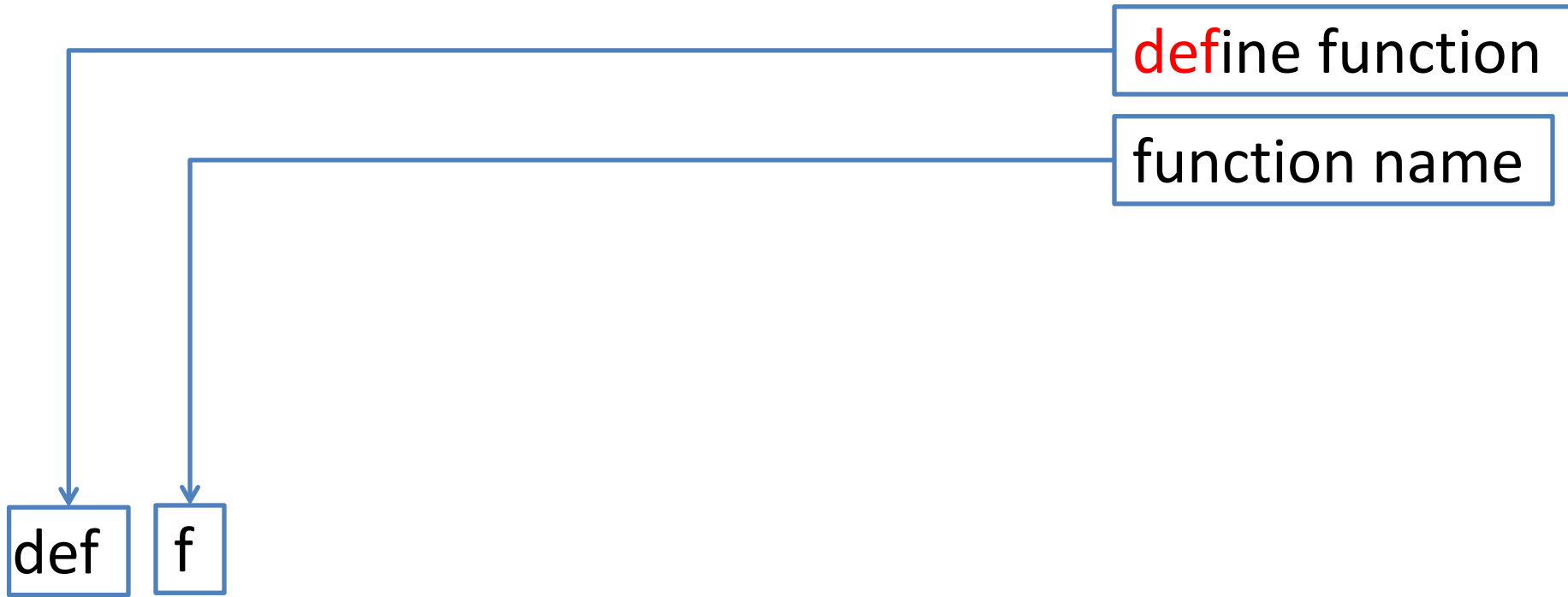
def

```
graph TD; A[define function] --> B[def];
```

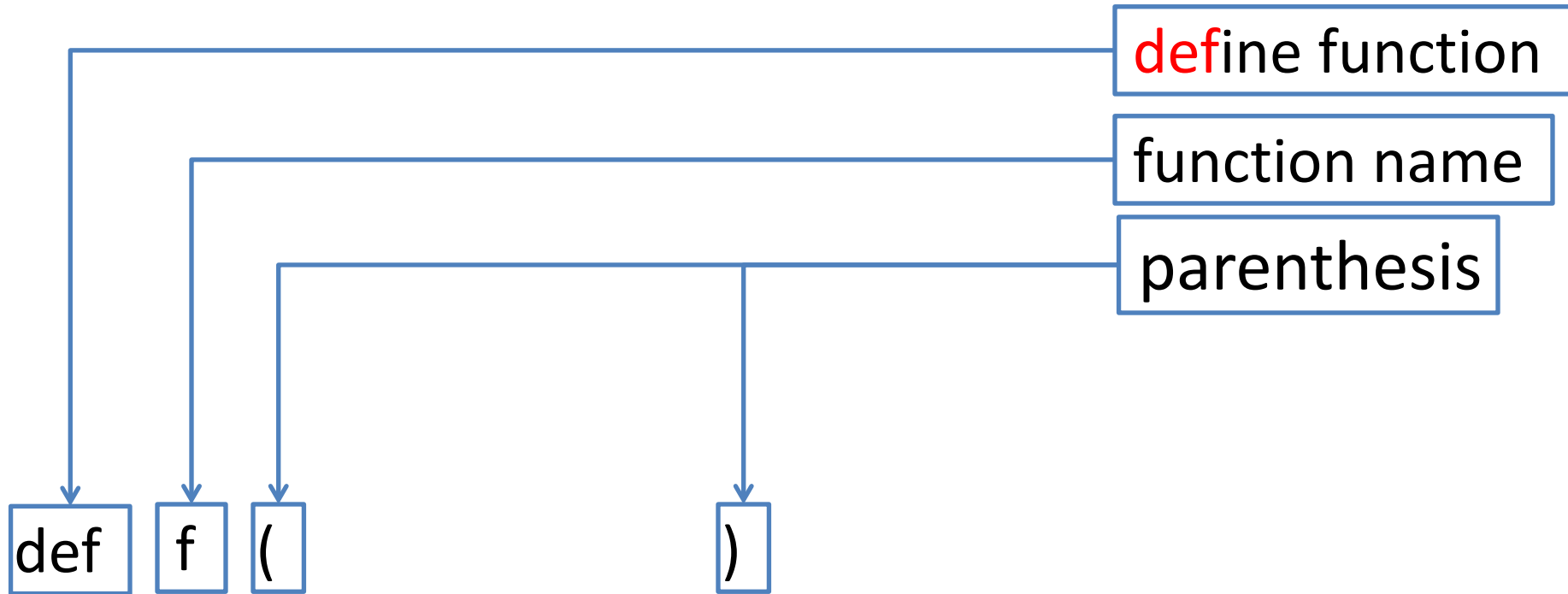
A diagram illustrating the relationship between the two terms. A horizontal blue line connects the 'define function' box on the right to a vertical blue line on the left. The vertical line ends in an arrow pointing down to the 'def' box.



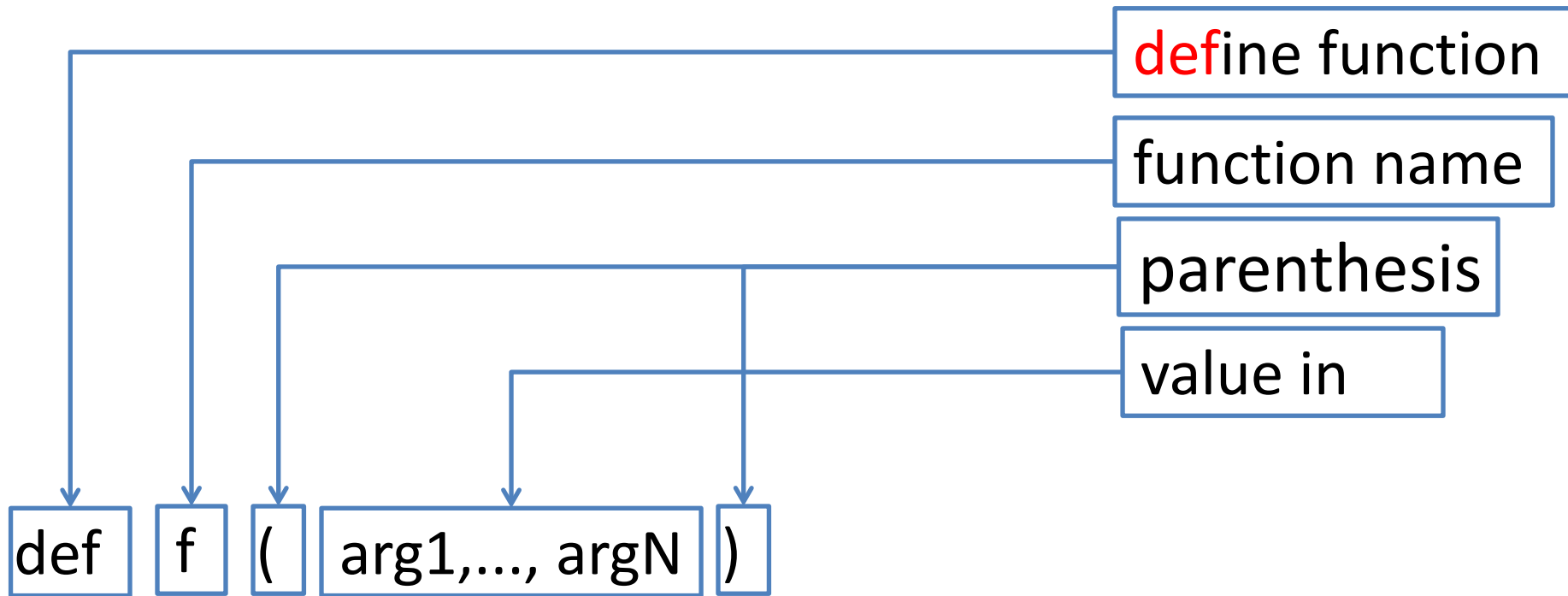
# Defining functions



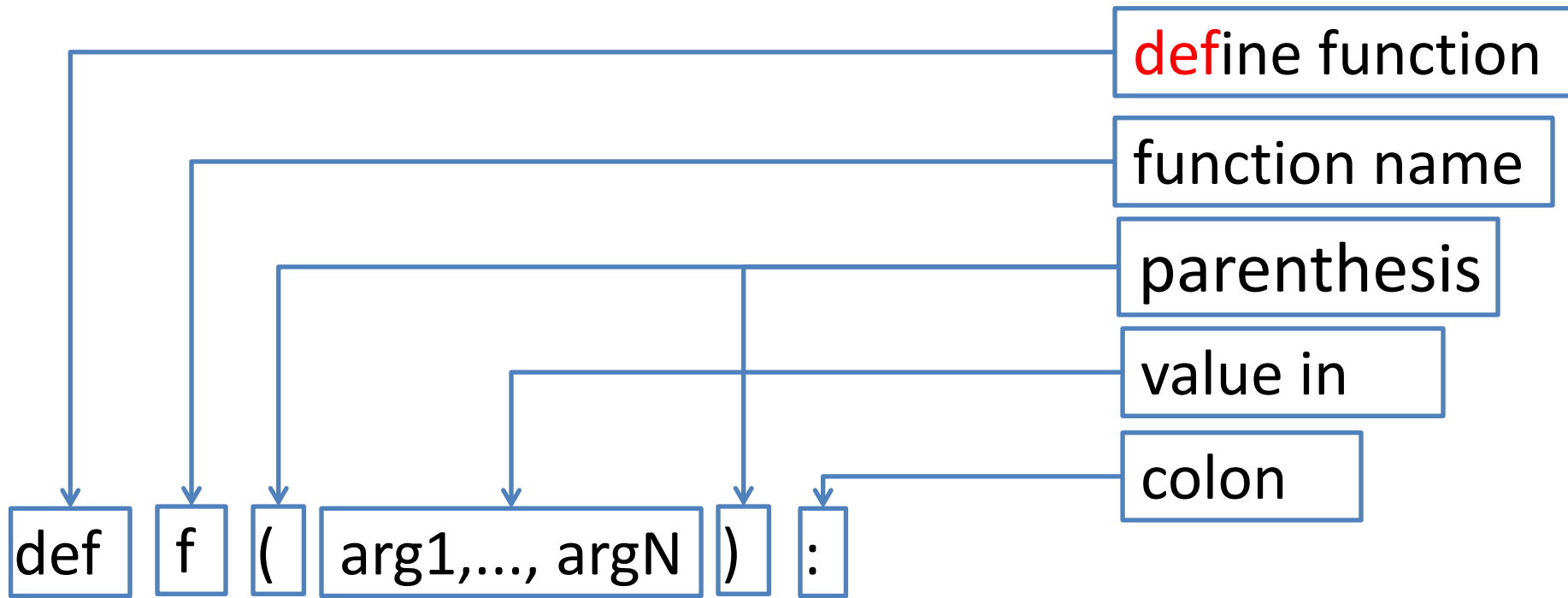
# Defining functions



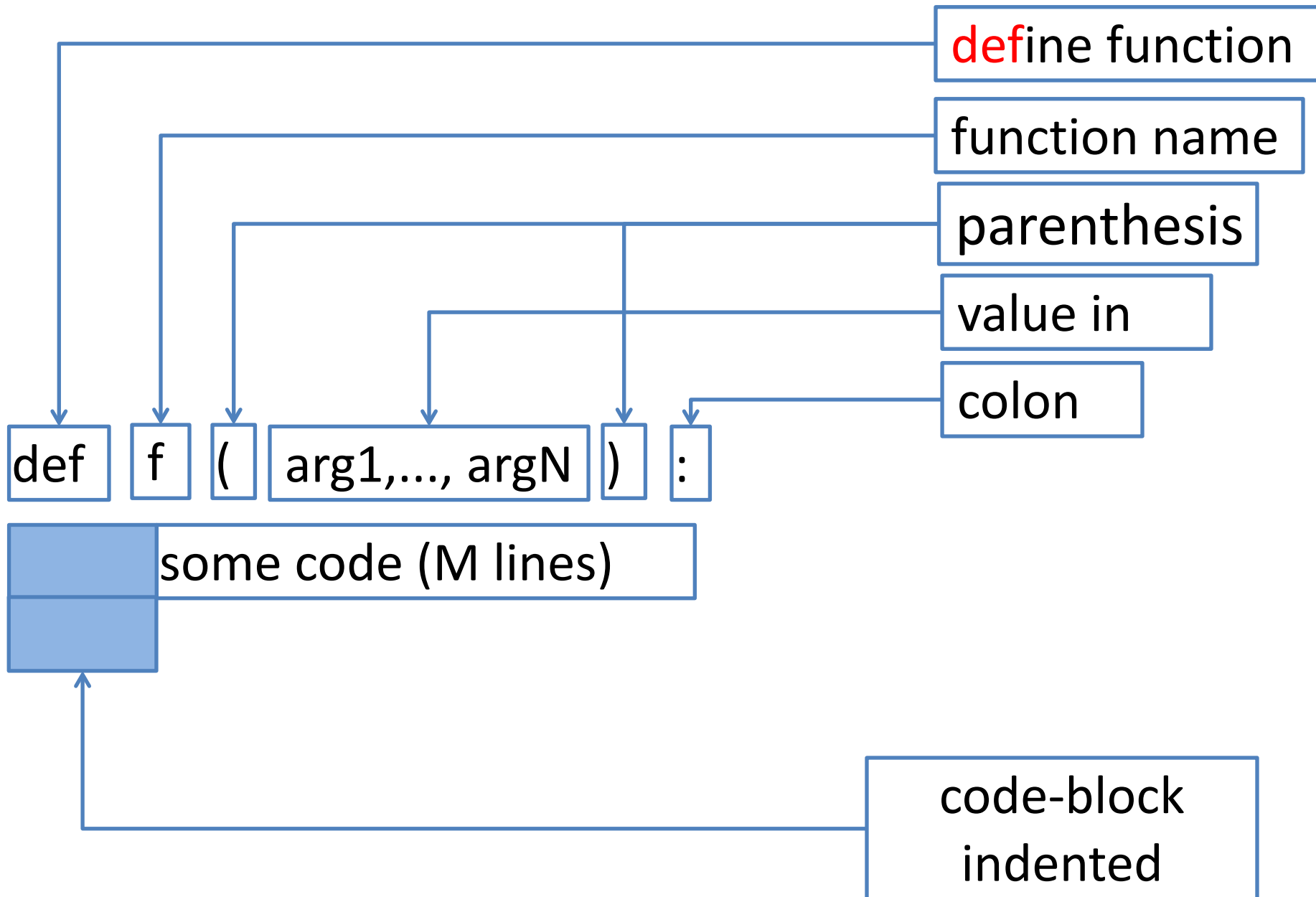
# Defining functions



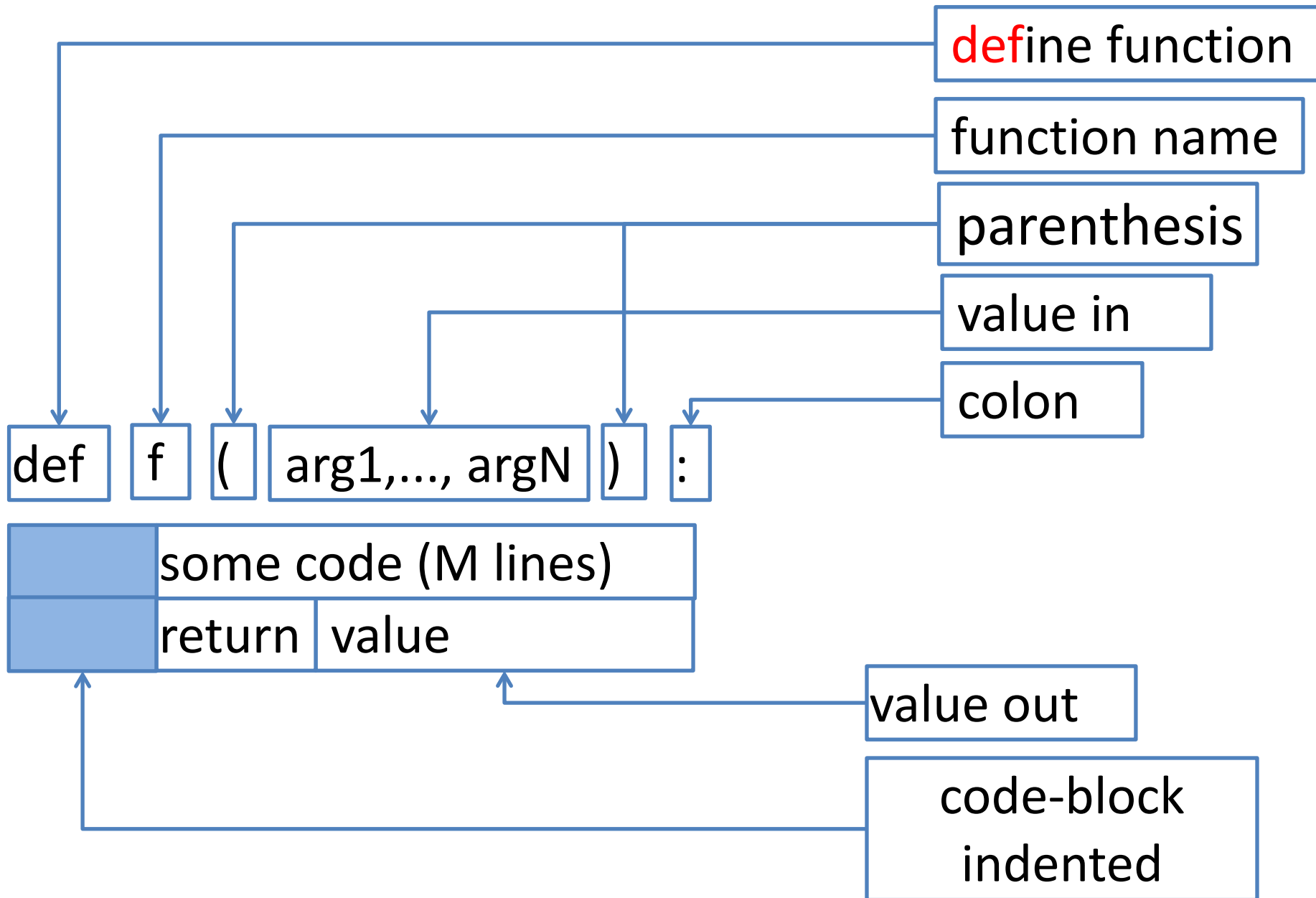
# Defining functions

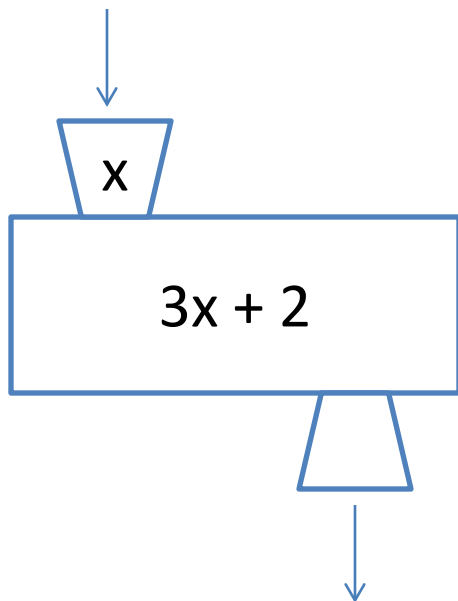


# Defining functions



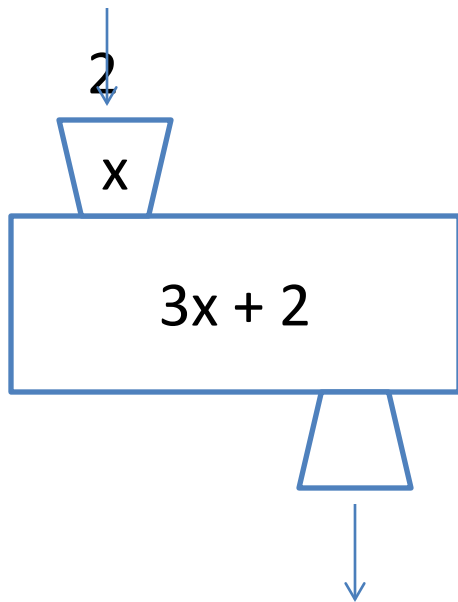
# Defining functions





math

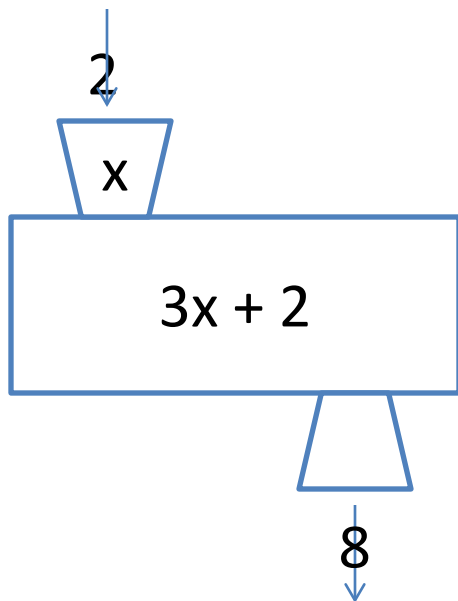
$$f(x) = 3x + 2$$



math

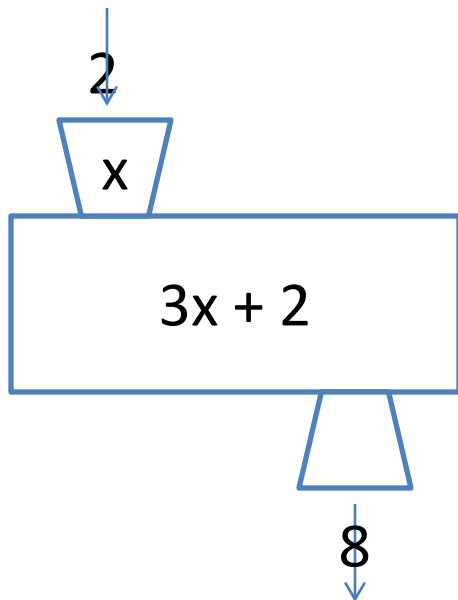
$$f(x) = 3x + 2$$





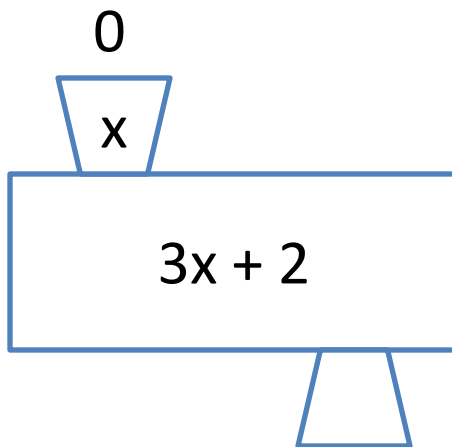
math

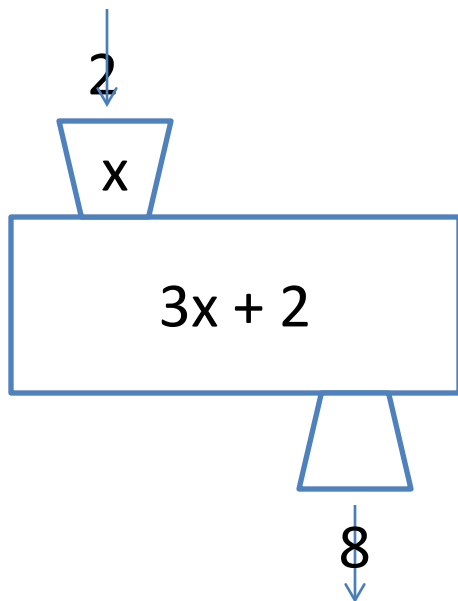
$$f(x) = 3x + 2$$



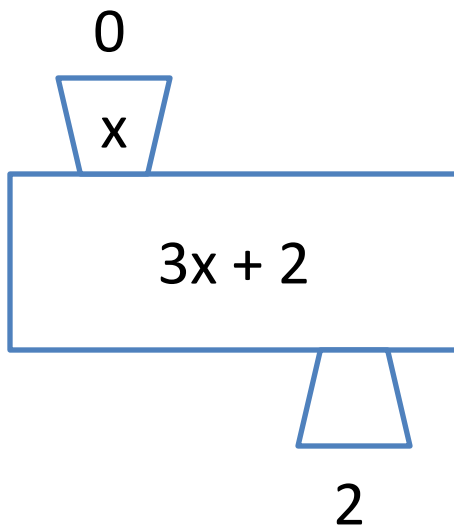
math

$$f(x) = 3x + 2$$

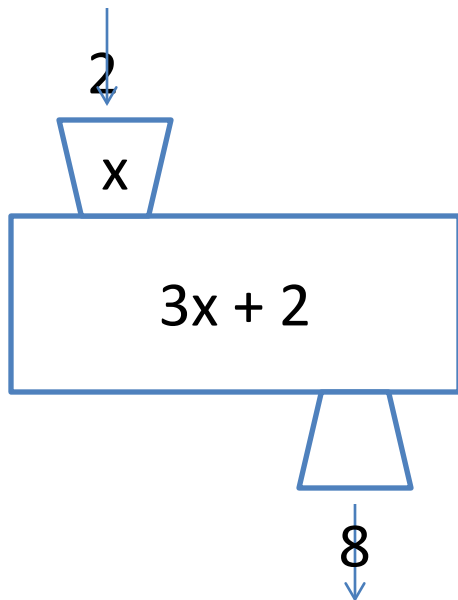




math

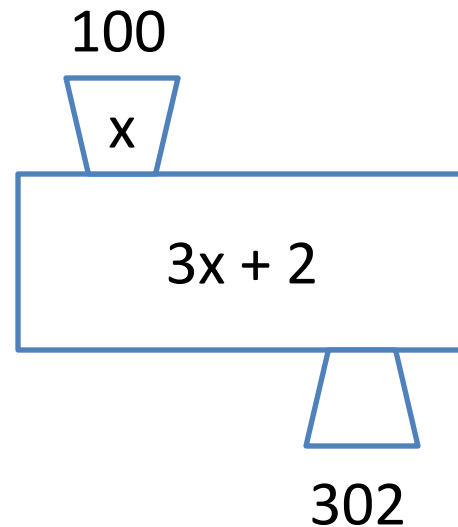
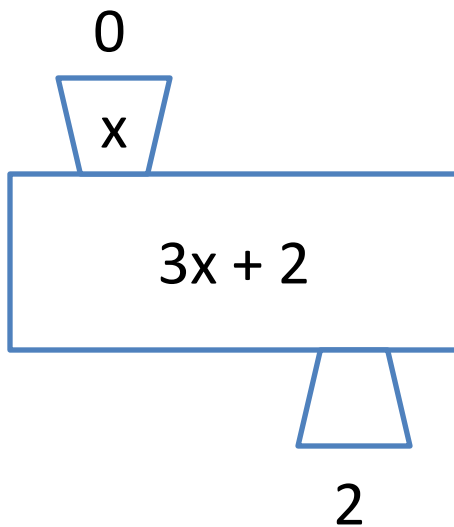


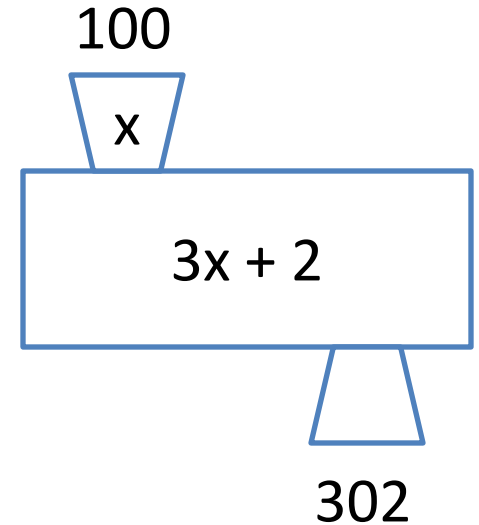
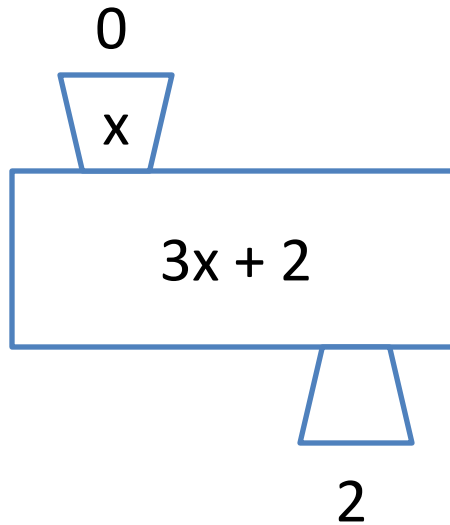
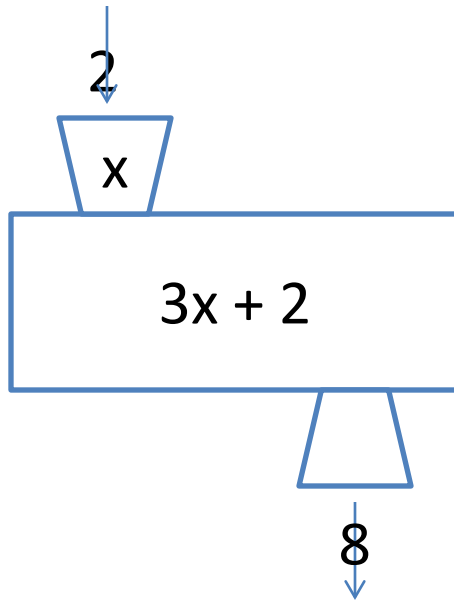
$$f(x) = 3x + 2$$



math

$$f(x) = 3x + 2$$





math

$$f(x) = 3x + 2$$

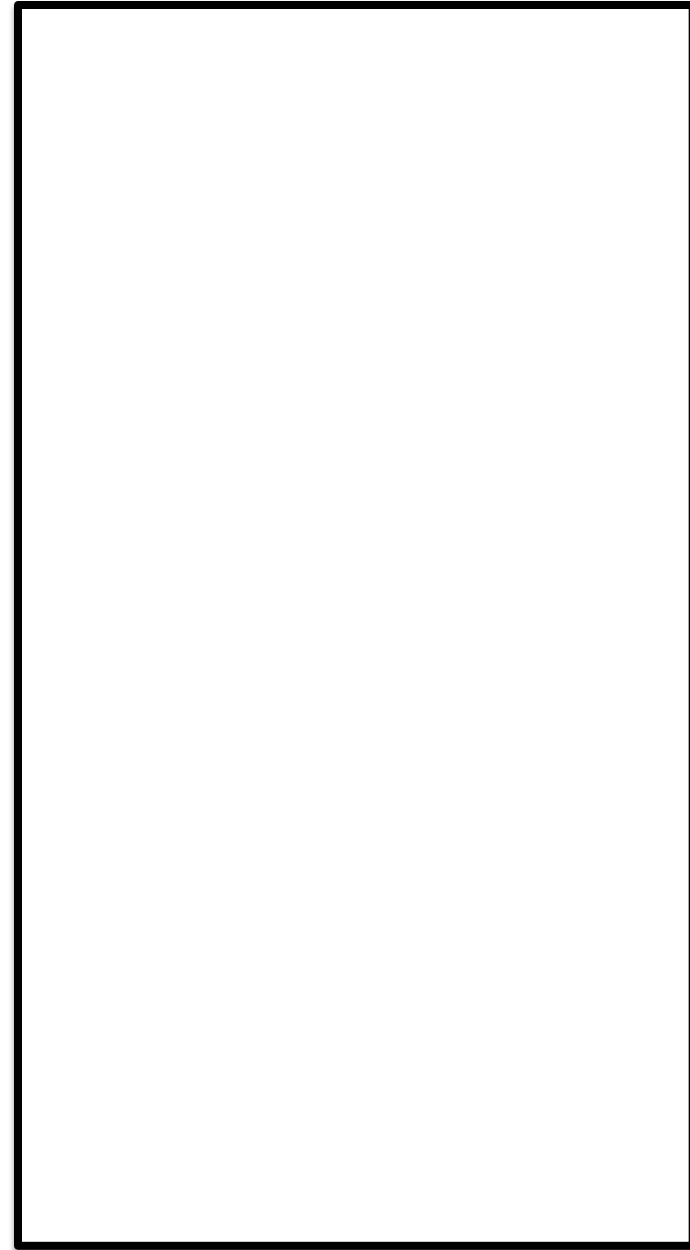
python

```
def f(x):  
    return 3*x+2
```

# defining functions

built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`



# defining functions

## built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

```
>>> abs(-9)
9
>>> max(2, 4)
4
>>> lst = [2,3,4,5]
>>> len(lst)
4
>>> sum(lst)
14
>>> print()

>>>
```

# defining functions

built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

նոր function-ներ սահմանվում են  
**def** keyword-ով

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2,3,4,5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>>
```




# defining functions

built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

նոր function-ներ սահմանվում են  
**def** keyword-ով

def: define keyword



```
def f(x):  
    res = x**2 + 10  
    return res
```

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2,3,4,5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>>
```

# defining functions

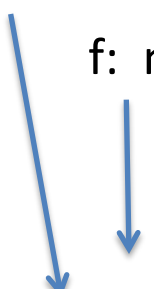
built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

նոր function-ներ սահմանվում են  
**def** keyword-ով

def: define keyword

f: name of function



```
def f(x):  
    res = x**2 + 10  
    return res
```

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2,3,4,5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>>
```

# defining functions

built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

նոր function-ներ սահմանվում են  
**def** keyword-ով

def: define keyword

f: name of function

x: input argument

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2,3,4,5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>>
```

# defining functions

built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

նոր function-ներ սահմանվում են  
**def** keyword-ով

def: define keyword

f: name of function

x: input argument

```
def f(x):  
    res = x**2 + 10  
    return res
```

return: function output

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2,3,4,5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>>
```

# defining functions

built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

նոր function-ներ սահմանվում են  
**def** keyword-ով

def: define keyword

f: name of function

x: input argument

```
def f(x):  
    res = x**2 + 10  
    return res
```

return: function output

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2,3,4,5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>> def f(x):  
        res = 2*x + 10  
        return res  
  
>>>
```

# defining functions

built-in functions:

- `abs()`, `max()`, `len()`, `sum()`, `print()`

նոր function-ներ սահմանվում են  
**def** keyword-ով

def: define keyword

f: name of function

x: input argument

```
def f(x):  
    res = x**2 + 10  
    return res
```

return: function output

```
>>> abs(-9)  
9  
>>> max(2, 4)  
4  
>>> lst = [2,3,4,5]  
>>> len(lst)  
4  
>>> sum(lst)  
14  
>>> print()  
  
>>> def f(x):  
        res = 2*x + 10  
        return res  
  
>>> f(1)  
12  
>>> f(3)  
16  
>>> f(0)  
10
```

# print() vs return

```
def f(x):  
    res = x**2 + 10  
    return res  
  
y = f(5)  
print(y)
```

Function returns value of res.

Can be used later

(Ֆունկցիան վերադարձնում է արժեք, որը կօգտագործվի հետո)

# print() vs return

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
y = f(5)  
print(y)
```

Function returns value of res.

Can be used later

(Ֆունկցիան վերադարձնում է արժեք, որը կօգտագործվի հետո)

```
def f(x):  
    res = x**2 + 10  
    print(res)
```

```
f(5)
```

Function prints value of res.

Does not return anything.

(Ֆունկցիան տպում է արժեք, բայց ոչինչ չի վերադարձնում)

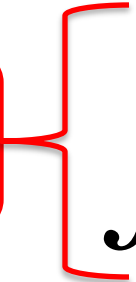


Մահմանէլ vs Կանչէլ

$$f(x) = x^2 + 10$$

# Սահմանել vs Կանչել

function definition  
(սահմանում)


$$f(x) = x^2 + 10$$

# Մահմանել vs Կանչել

function definition  
(սահմանում)

$$f(x) = x^2 + 10$$

$$f(5)$$

# Մահմանել vs Կանչել

function definition  
(սահմանում)

$$f(x) = x^2 + 10$$

function call  
(կանչ)

$$f(5)$$

# Մահմանել vs Կանչել

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
f(5)
```

module1.py

# Մաիմանել vs Կանչել

function definition

$$f(x) = x^2 + 10$$

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
f(5)
```

module1.py

# Մահմանել vs Կանչել

function definition

$$f(x) = x^2 + 10$$

function call

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
f(5)
```

module1.py

# Մահմանել vs Կանչել

function definition

```
def f(x):  
    res = x**2 + 10  
    return res
```

function call

```
print(f(5))
```

module1.py



# Մահմանել vs Կանչել

Function Definition

```
def f(x):  
    res = x**2 + 10  
    return res
```

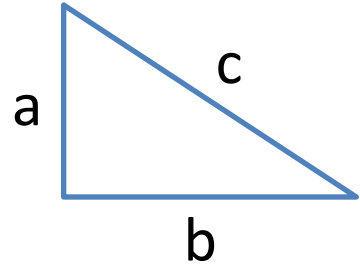
Function call

```
y = f(5)  
print(y)
```

module1.py

# Exercise 2A

Գրեք ծրագիր որը հաշվում է  
ուղղանկյուն եռանկյան  
ներքնաձիգի երկարությունը



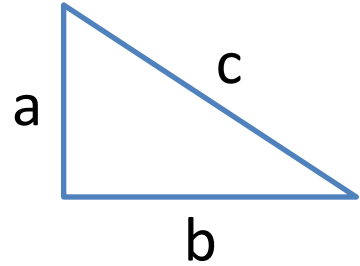
```
import math
```

```
c = math.sqrt(a**2 + b**2)  
print(c)
```

# Exercise 2B

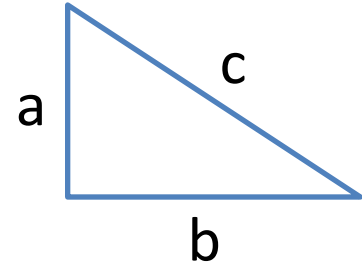
Գրեք function, որը հաշվում է  
ուղղանկյուն եռանկյան  
ներքնաձիգի երկարությունը

```
>>> hyp(3,4)  
5.0  
>>>
```



# Exercise 2B

Գրեք function, որը հաշվում է  
ուղղանկյուն եռանկյան  
ներքնաձիգի երկարությունը



```
>>> hyp(3,4)
5.0
>>>
```

```
import math
```

```
def hyp(a, b):
    res = math.sqrt(a**2 + b**2)
    return res
```

```
c = hyp(3,4))
print(c)
```

ex2.py

# Exercise 3

Գրեք function hello() որը

- ունի մեկ արգումենտ name
- տպում է welcome message հետևյալ կերպ

```
>>> hello('Julie')  
Welcome, Julie!  
>>>
```

# Exercise 3

Գրեք function hello() որը

- ունի մեկ արգումենտ name
- տպում է welcome message հետևյալ կերպ

```
>>> hello('Julie')
Welcome, Julie!
>>>
```

```
def hello(name):
    line = 'Welcome, ' + name + '!'
    print(line)
```

```
hello('Julie')
```

ex3.py

# Exercise 3

Գրեք function hello() որը

- ունի մեկ արգումենտ name
- տպում է welcome message հետևյալ կերպ

```
>>> hello('Julie')  
Welcome, Julie!  
>>>
```

Function Definition

```
def hello(name):  
    line = 'Welcome, ' + name + '!'  
    print(line)
```

```
hello('Julie')
```

ex3.py

# Exercise 3

Գրեք function hello() որը

- ունի մեկ արգումենտ name
- տպում է welcome message հետևյալ կերպ

```
>>> hello('Julie')
Welcome, Julie!
>>>
```

Function Definition

```
def hello(name):
    line = 'Welcome, ' + name + '!'
    print(line)
```

Function call

```
hello('Julie')
```

ex3.py



# Comments and docstrings

## code documentation

- developer-ը նվ գրում է code-ը  
հասկանում է code-ը
- user-ը նվ օգտագործում է code-ը  
գիտի ինչ է անում code-ը

# Comments and docstrings

## code documentation

- developer-ը նվ գրում է code-ը հասկանում է code-ը
- user-ը նվ օգտագործում է code-ը գիտի ինչ է անում code-ը

```
def f(x):  
    res = x**2 + 10  
    return res
```

# Comments and docstrings

## code documentation

- developer-ը նվ գրում է code-ը հասկանում է code-ը
- user-ը նվ օգտագործում է code-ը գիտի ինչ է անում code-ը

```
def f(x):  
    res = x**2 + 10  
    return res
```

```
>>> help(f)
```

```
Help on function f in module  
__main__:
```

```
f(x)
```

# Comments and docstrings

## code documentation

- developer-ը նվ գրում է code-ը հասկանում է code-ը
- user-ը նվ օգտագործում է code-ը գիտի ինչ է անում code-ը

```
def f(x):  
    res = x**2 + 10  
    return res
```

## Docstring

```
def f(x):  
    'returns x**2 + 10'  
    res = x**2 + 10 #calculate res  
    return res
```

```
>>> help(f)  
Help on function f in module  
__main__:
```

```
f(x)
```

# Comments and docstrings

## code documentation

- developer-ը ունի գրում է code-ը հասկանում է code-ը
- user-ը ունի օգտագործում է code-ը գիտի ինչ է անում code-ը

```
def f(x):  
    res = x**2 + 10  
    return res
```

## Docstring

```
def f(x):  
    'returns x**2 + 10'  
    res = x**2 + 10 #calculate res  
    return res
```

```
>>> help(f)  
Help on function f in module  
__main__:
```

```
f(x)
```

```
>>> def f(x):  
        'returns x**2 + 10'  
        res = x**2 + 10  
        return res
```

```
>>> help(f)  
Help on function f in module  
__main__:
```

```
f(x)  
    returns x**2 + 10
```

```
>>>
```

# Multi-line strings : Documentation

multiline string

```
s = """ """
```

# Multi-line strings : Documentation

```
multiline string  
s = """ """
```

```
def square(num):  
    """ (number) -> number  
    Returns the square of  
        num.  
    >>> square(3)  
    9  
    """  
  
    return num**2  
  
print(square(2))
```

# Exercise 4A

Գրեք ծրագիր որը հաշվում է **lst=[1,2,3,4,5]**  
անդամների գումարը



# Exercise 4A

Գրեք ծրագիր որը հաշվում է **lst=[1,2,3,4,5]**  
անդամների գումարը

```
total = 0
for number in lst:
    total = ...

print(total)
```

ex4.py

# Exercise 4B

Գրեք `summa()` function որը վերցնում է list որպես input և վերադարձնում է list-ի անդամների գումարը

```
>>> lst=[1,2,3,4,5]
>>> summa(lst)
15
```

# Exercise 4B

Գրեք `summa()` function որը վերցնում է list որպես input և վերադարձնում է list-ի անդամների գումարը

```
>>> lst=[1,2,3,4,5]
>>> summa(lst)
15
```

```
def summa(lst):
    """ Calculate sum of list elements"""
    total = 0
    for number in lst:
        total = total + number
    return total
```

```
y = summa([1,2,3,4,5])
print(y)
```

ex4.py

# objects and classes

Python-ում ամեն ինչ

int, float, string, list, ...

object է



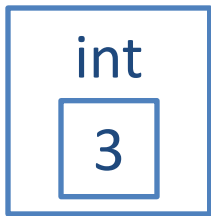
# objects and classes

Python-ում ցանկացած արժեք, լինի  
դա **int, float, string** կամ **list** պահվում  
է հիշողության մեջ որպես **object**



# objects and classes

Python-ում ցանկացած արժեք, լինի  
դա **int**, **float**, **string** կամ **list** պահվում  
է հիշողության մեջ որպես **object**



```
>>> a = 3
>>>
```



# objects and classes

Python-ում ցանկացած արժեք, լինի  
դա **int**, **float**, **string** կամ **list** պահվում  
է հիշողության մեջ որպես **object**



```
>>> a = 3
>>> b = 3.0
>>>
```



# objects and classes

Python-ում ցանկացած արժեք, լինի  
դա **int**, **float**, **string** կամ **list** պահվում  
է հիշողության մեջ որպես **object**



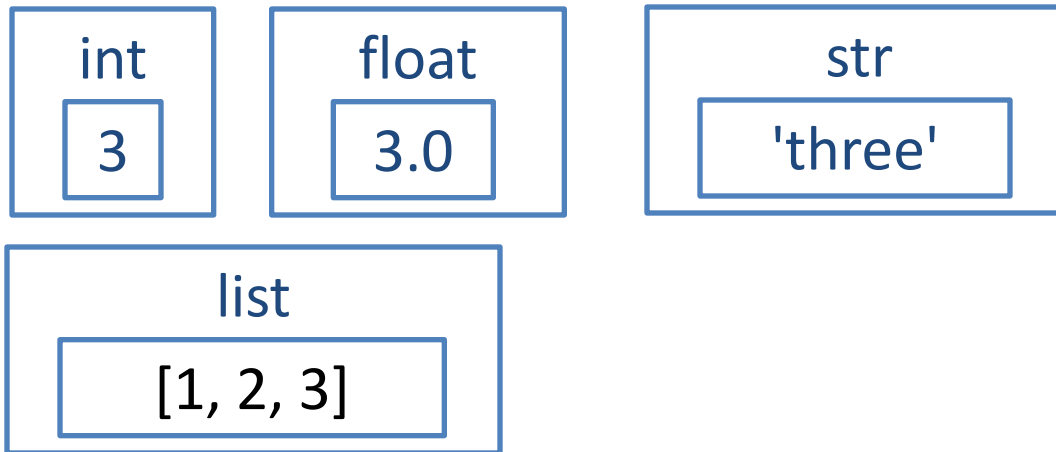
```
>>> a = 3
>>> b = 3.0
>>> c = 'three'
>>>
```





# objects and classes

Python-ում ցանկացած արժեք, լինի  
դա **int**, **float**, **string** կամ **list** պահվում  
է հիշողության մեջ որպես **object**



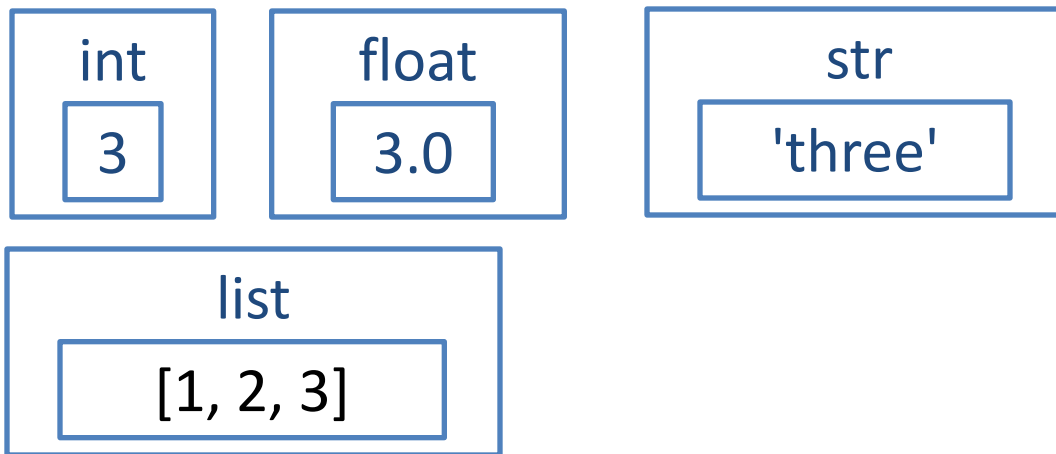
```
>>> a = 3
>>> b = 3.0
>>> c = 'three'
>>> d = [1, 2, 3]
>>>
```



# objects and classes

Python-ում ցանկացած արժեք, լինի դա **int**, **float**, **string** կամ **list** պահվում է հիշողության մեջ որպես **object**

Ամեն **object** ունի **value** և **type**;

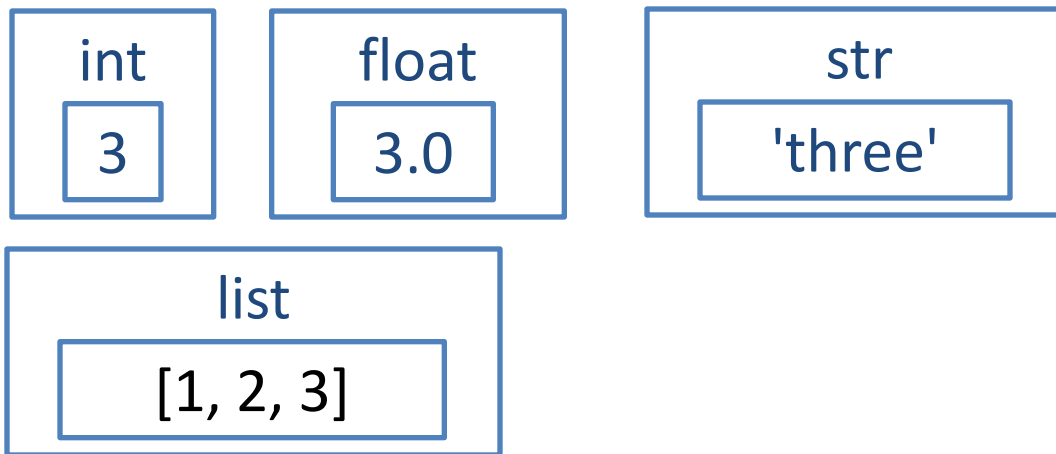


```
>>> a = 3
>>> b = 3.0
>>> c = 'three'
>>> d = [1, 2, 3]
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'list'>
>>>
```

# objects and classes

Python-ում ցանկացած արժեք, լինի դա **int**, **float**, **string** կամ **list** պահվում է հիշողության մեջ որպես **object**

Ամեն **object** ունի **value** և **type**;



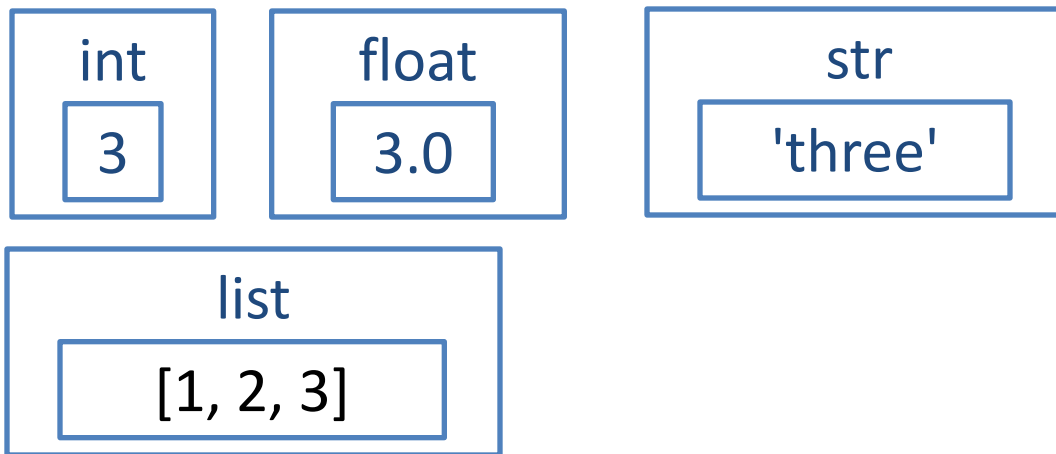
```
>>> a = 3
>>> b = 3.0
>>> c = 'three'
>>> d = [1, 2, 3]
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'list'>
>>>
```

Terminology: **object X** is of **type int** =  
**object X** belongs to **class int** (պատկանում է int class-ին)

# objects and classes

Python-ում ցանկացած արժեք, լինի դա **int**, **float**, **string** կամ **list** պահվում է հիշողության մեջ որպես **object**

Ամեն **object** ունի **value** և **type**;



```
>>> a = 3
>>> b = 3.0
>>> c = 'three'
>>> d = [1, 2, 3]
>>> type(a)
<class 'int'>
>>> type(b)
<class 'float'>
>>> type(c)
<class 'str'>
>>> type(d)
<class 'list'>
>>> a = []
>>> type(a)
<class 'list'>
```

Terminology: **object X** is of **type int** =  
**object X** belongs to **class int** (պատկանում է int class-ին)



# Memory Model

```
>>> x = 23.0
```

```
>>> id(x)
```

```
31947344
```

```
>>> x
```

```
23.0
```

Մեր memory model-ում x-ը պարունակում է float object-ի հասցեն: id1-ը

id1 -> id(x)->31947344



# Memory Model

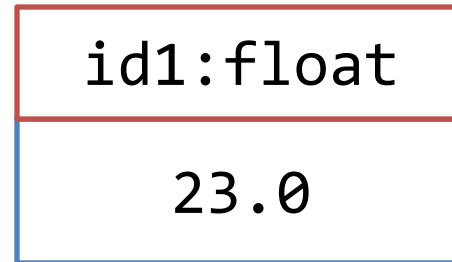
```
>>> x = 23.0
```

```
>>> id(x)
```

```
31947344
```

```
>>> x
```

```
23.0
```



Մեր memory model-ում x-ը պարունակում է float object-ի հասցեն: id1-ը

id1 -> id(x)->31947344



# Memory Model

```
>>> x = 23.0
```

```
>>> id(x)
```

```
31947344
```

```
>>> x
```

```
23.0
```

x

id1

id1:float

23.0

Մեր memory model-ում x-ը պարունակում է float object-ի հասցեն: id1-ը

id1 -> id(x)->31947344

# Memory Model

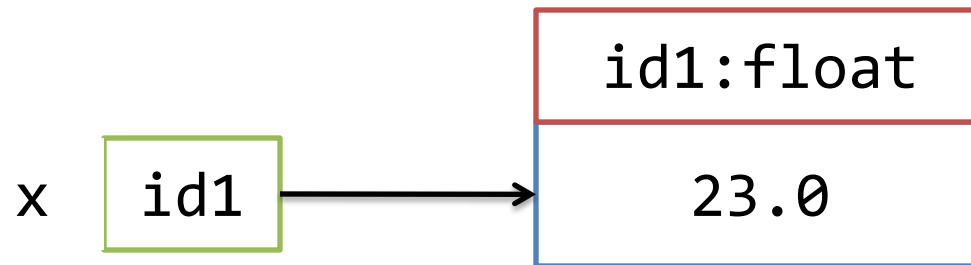
```
>>> x = 23.0
```

```
>>> id(x)
```

```
31947344
```

```
>>> x
```

```
23.0
```



Մեր memory model-ում `x`-ը պարունակում է float object-ի հասցեն: `id1`-ը

`id1 -> id(x) -> 31947344`



# Memory Model

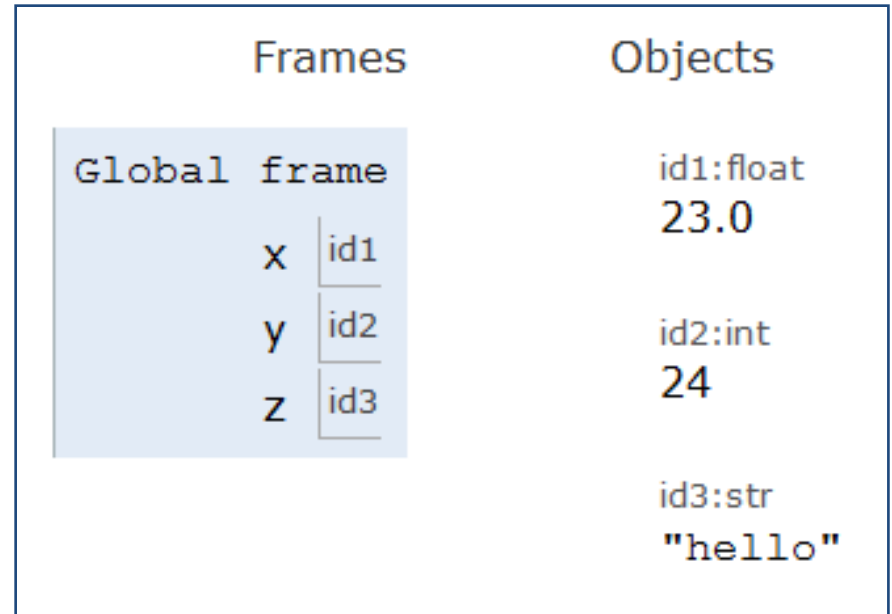
```
>>> x = 23.0
```

```
>>> y = 24
```

```
>>> z = "hello"
```

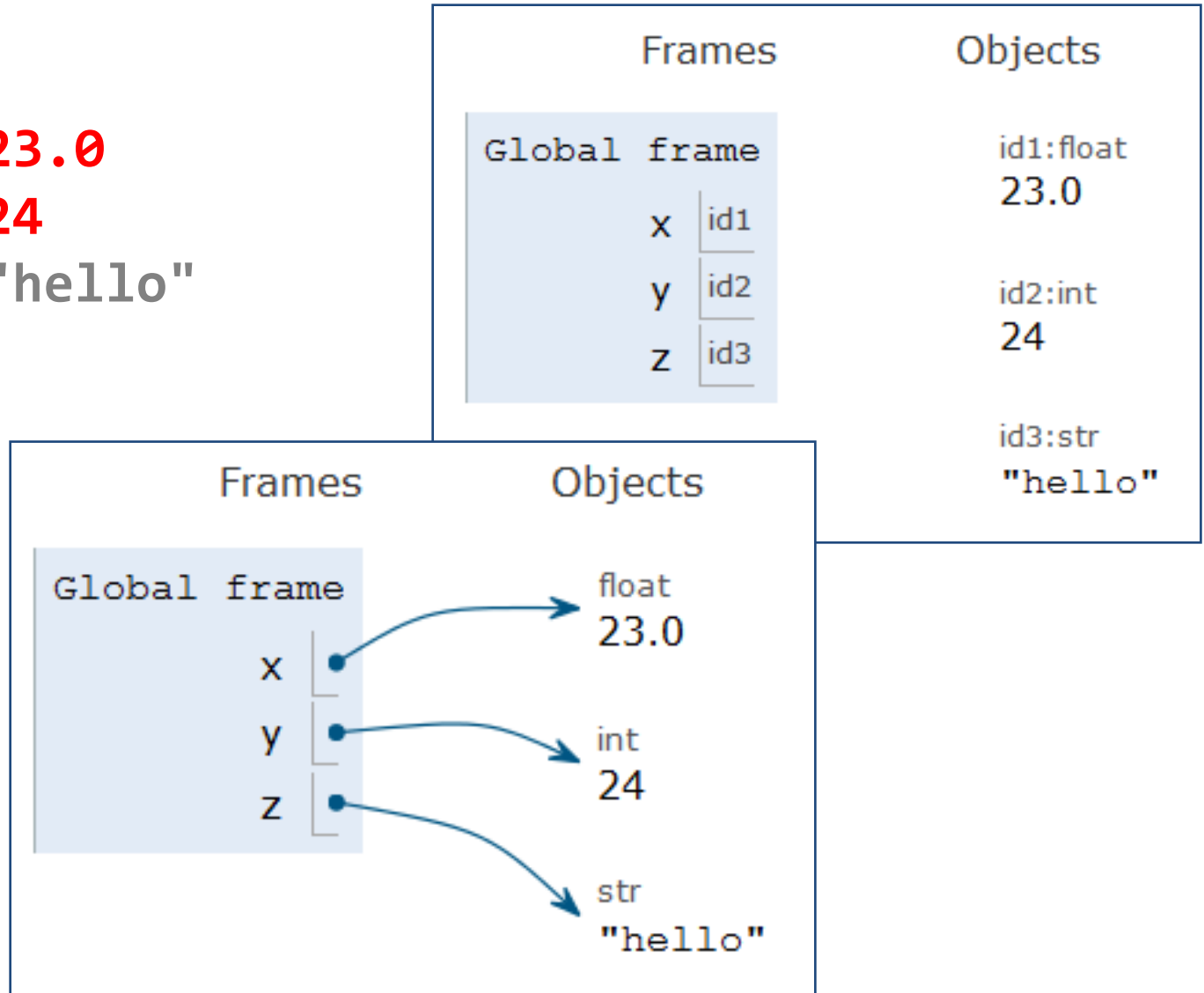
# Memory Model

```
>>> x = 23.0
>>> y = 24
>>> z = "hello"
```



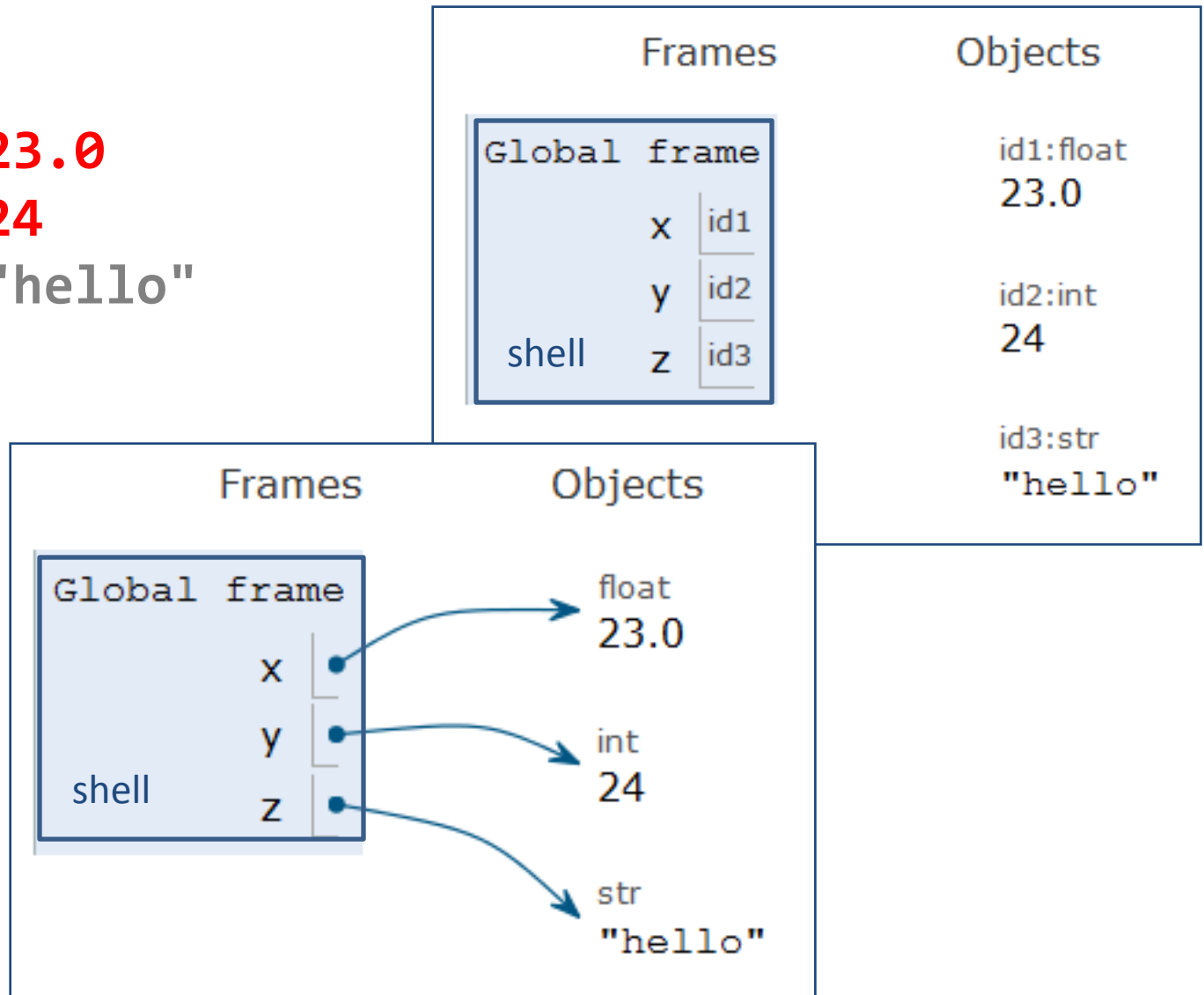
# Memory Model

```
>>> x = 23.0  
>>> y = 24  
>>> z = "hello"
```



# Memory Model

```
>>> x = 23.0  
>>> y = 24  
>>> z = "hello"
```

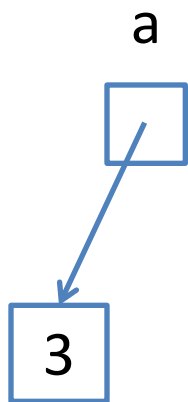


# Assignment statement “=”

փոփոխականը գոյություն չունի,  
մինչև վերագրումը

```
>>> a
Traceback (most
recent call last):
  File
"<pyshell#66>", line
1, in <module>
    a
NameError: name 'a'
is not defined
>>>
```

# Assignment statement "="



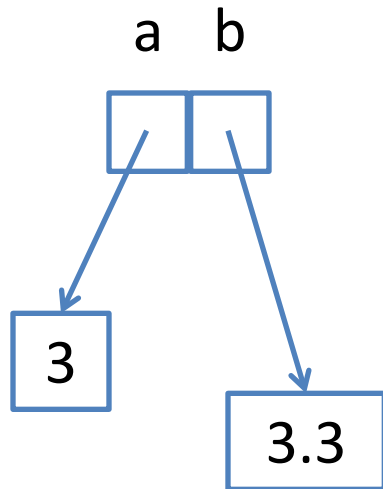
փոփոխականը գոյություն չունի,  
մինչև վերագրումը

`<variable> = <expression>`

**<expression>**-ը հաշվվում է և իր value-  
ն դրվում է որոշակի type-ի **object**-ի  
մեջ

```
>>> a
Traceback (most
recent call last):
  File
"<pyshell#66>", line
1, in <module>
    a
NameError: name 'a'
is not defined
>>> a = 3
```

# Assignment statement "="



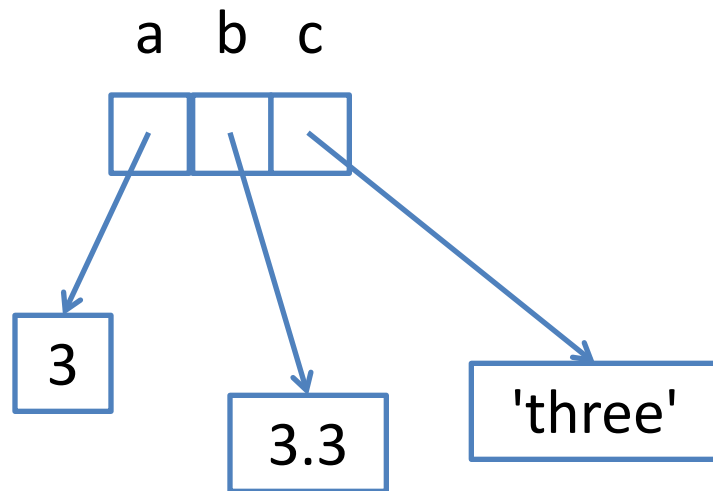
`<variable> = <expression>`

**<expression>**-ը հաշվվում է և իր value-ն դրվում է որոշակի type-ի **object**-ի մեջ

փոփոխականը գոյություն չունի, մինչև վերագրումը

```
>>> a
Traceback (most recent call last):
  File
"<pyshell#66>", line
1, in <module>
    a
NameError: name 'a'
is not defined
>>> a = 3
>>> b = 2 + 1.3
```

# Assignment statement "="



`<variable> = <expression>`

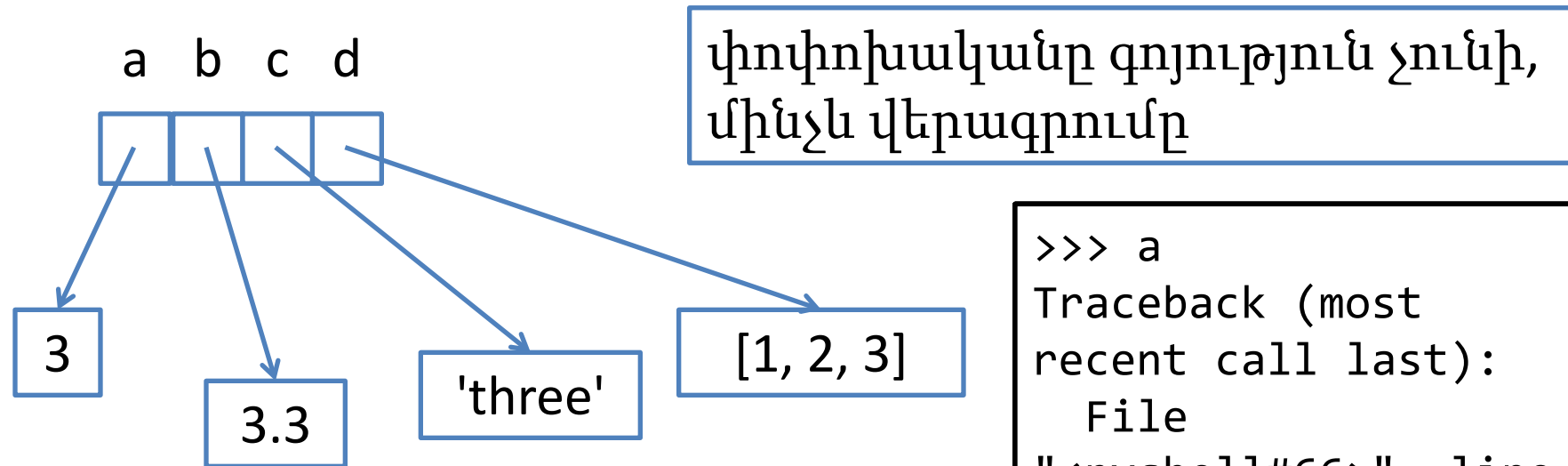
**<expression>**-ը հաշվվում է և իր value-ն դրվում է որոշակի type-ի **object**-ի մեջ

փոփոխականը գոյություն չունի, մինչև վերագրումը

```
>>> a
Traceback (most recent call last):
  File
"<pyshell#66>", line
1, in <module>
    a
NameError: name 'a'
is not defined
>>> a = 3
>>> b = 2 + 1.3
>>> c = 'three'
```



# Assignment statement "="



փոփոխականը գոյություն չունի,  
մինչև վերագրումը

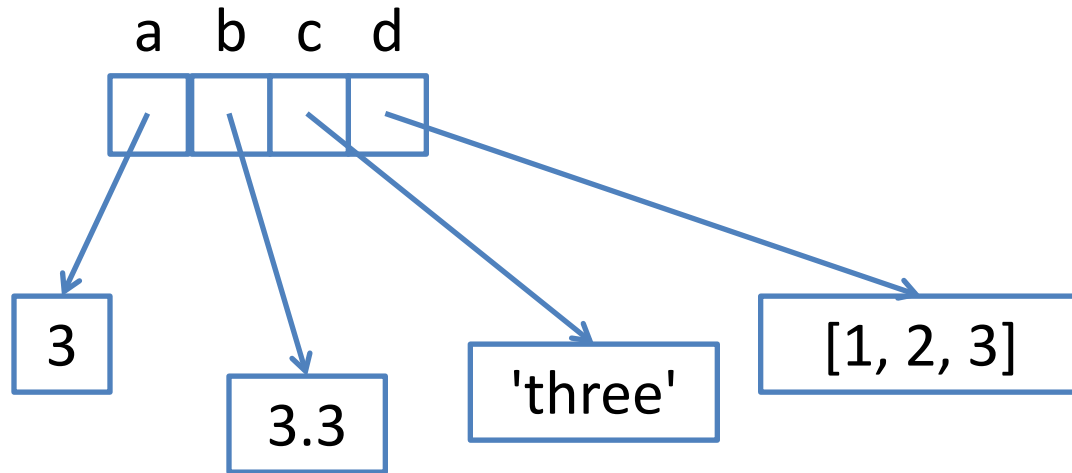
`<variable> = <expression>`

**<expression>**-ը հաշվվում է և իր value-ն դրվում է որոշակի type-ի **object**-ի մեջ

```
>>> a
Traceback (most recent call last):
  File
"<pyshell#66>", line
1, in <module>
    a
NameError: name 'a'
is not defined
>>> a = 3
>>> b = 2 + 1.3
>>> c = 'three'
>>> d = [1, 2] + [3]
```

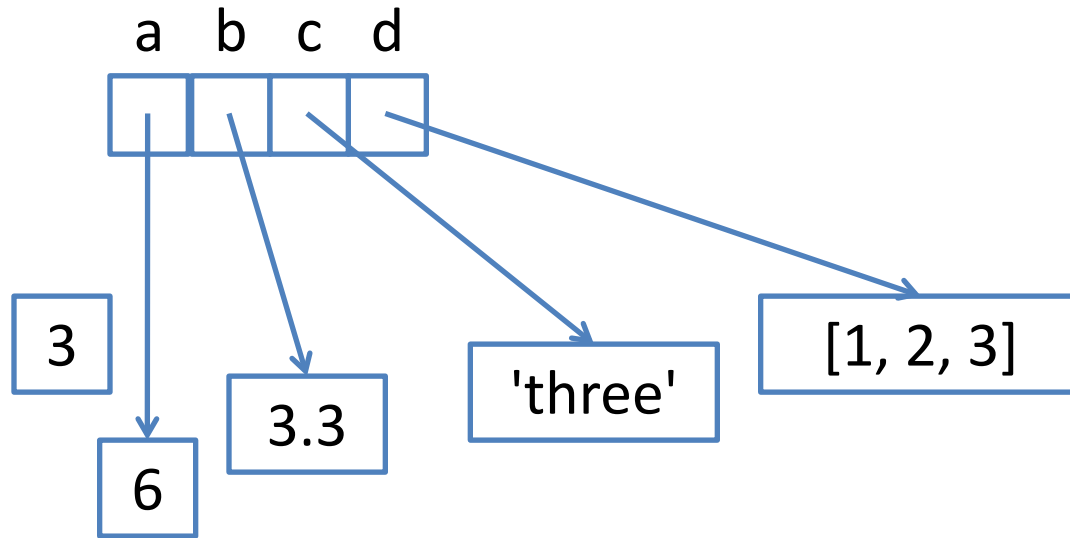


# Mutable and immutable types



```
>>> a  
3
```

# Mutable and immutable types



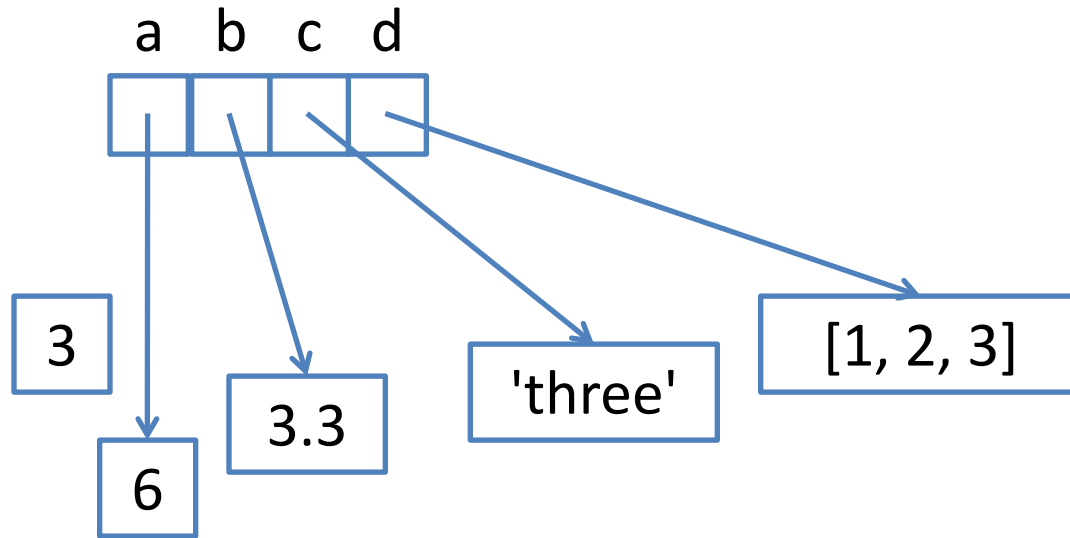
**object (3)** որին ցույց է տալիս a variable-ը չի փոխվում, փոխարենը ,

**a** ցույց է տալիս new **object (6)**

➤ Integers → **immutable**

```
>>> a
3
>>> a = 6
>>> a
6
```

# Mutable and immutable types

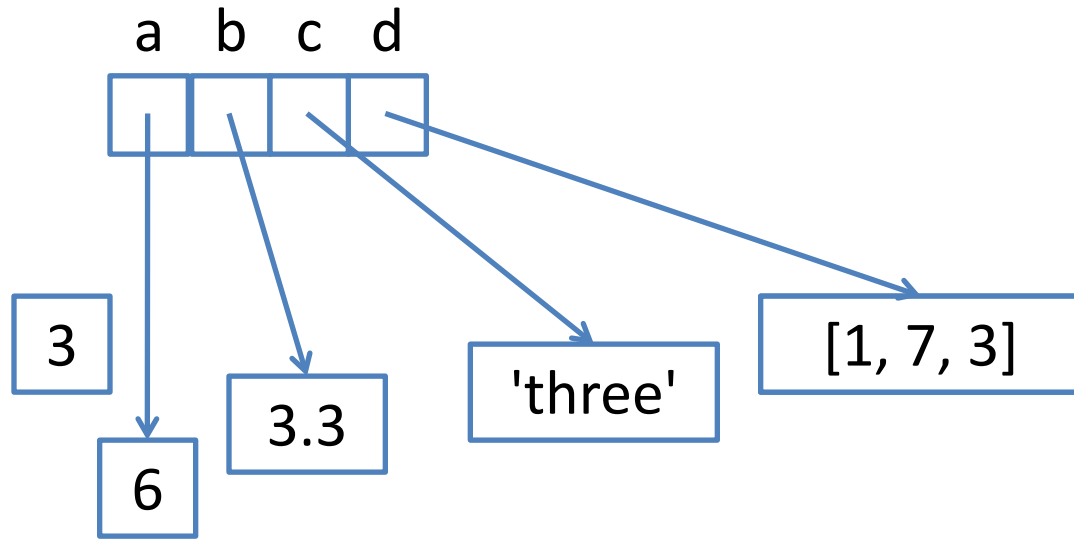


**object (3)** որին ցույց է տալիս a variable-ը չի փոխվում, փոխարենը ,  
**a** ցույց է տալիս new **object (6)**

➤ Integers → **immutable**

```
>>> a
3
>>> a = 6
>>> a
6
>>> d
[1, 2, 3]
```

# Mutable and immutable types



**object (3)** որին ցույց է տալիս a variable-ը չի փոխվում, փոխարենը ,  
**a** ցույց է տալիս new **object (6)**

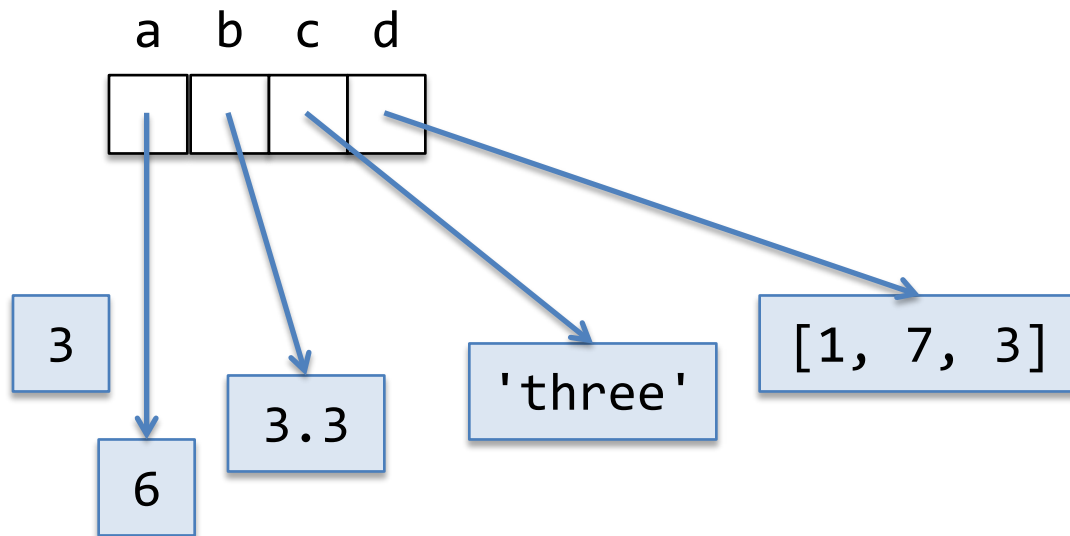
➤ Integers → **immutable**

The **object ([1, 2, 3])** որին ցույց է տալիս **d**-ն փոխվում է

➤ Lists → **mutable**

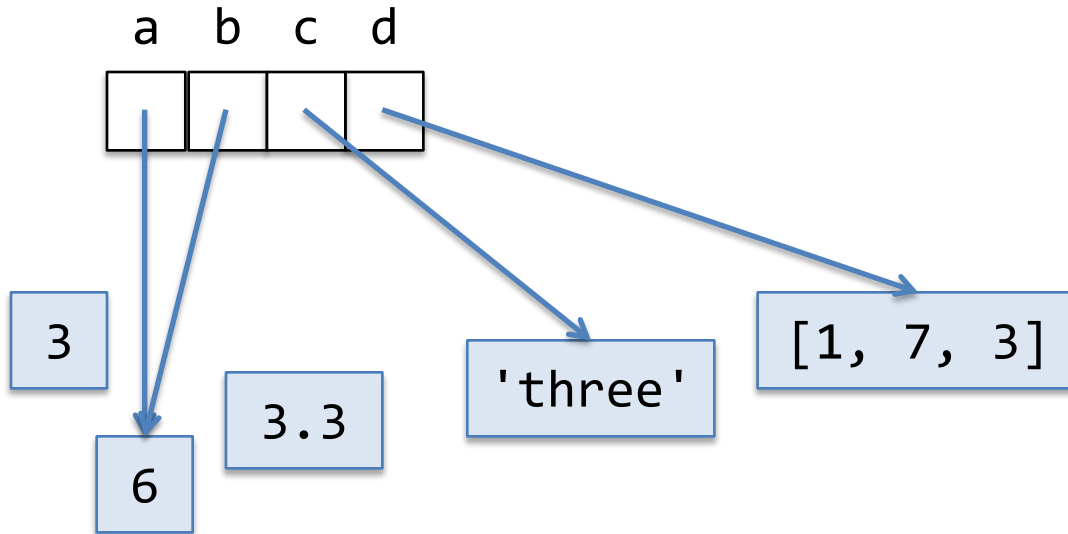
```
>>> a
3
>>> a = 6
>>> a
6
>>> d
[1, 2, 3]
>>> d[1] = 7
>>> d
[1, 7, 3]
```

# Assignment and mutability



```
>>> a
6
>>> b
3.3
```

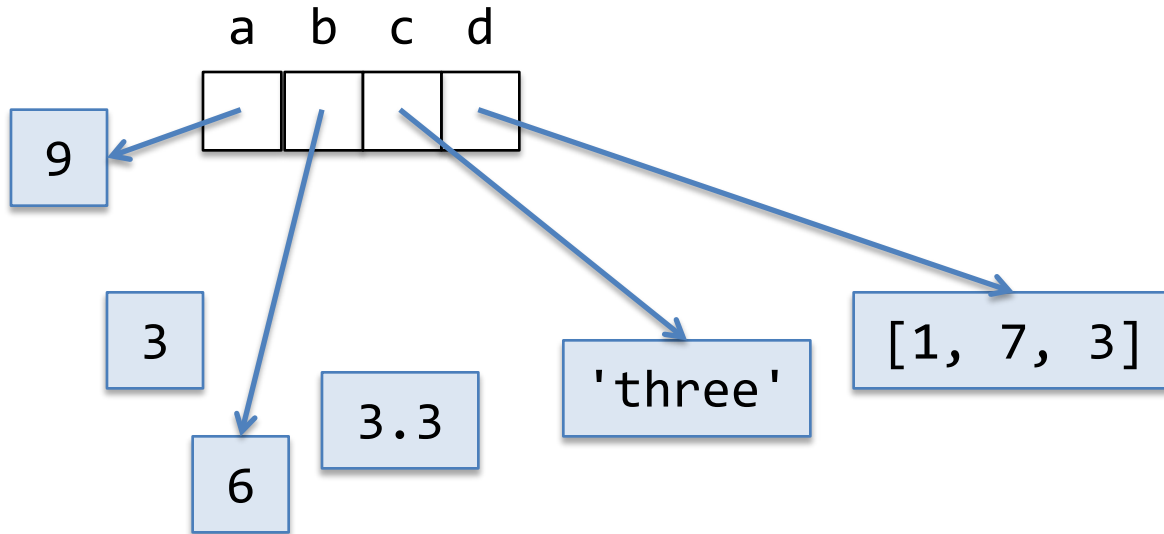
# Assignment and mutability



**a** and **b** ցույց են տալիս նույն int-ին

```
>>> a
6
>>> b
3.3
>>> b = a
>>> b
6
```

# Assignment and mutability



**a** ցույց է տալիս նոր **object (9)**;

**b** դեռ ցույց է տալիս հին **object (6)**

➤ integer → immutable

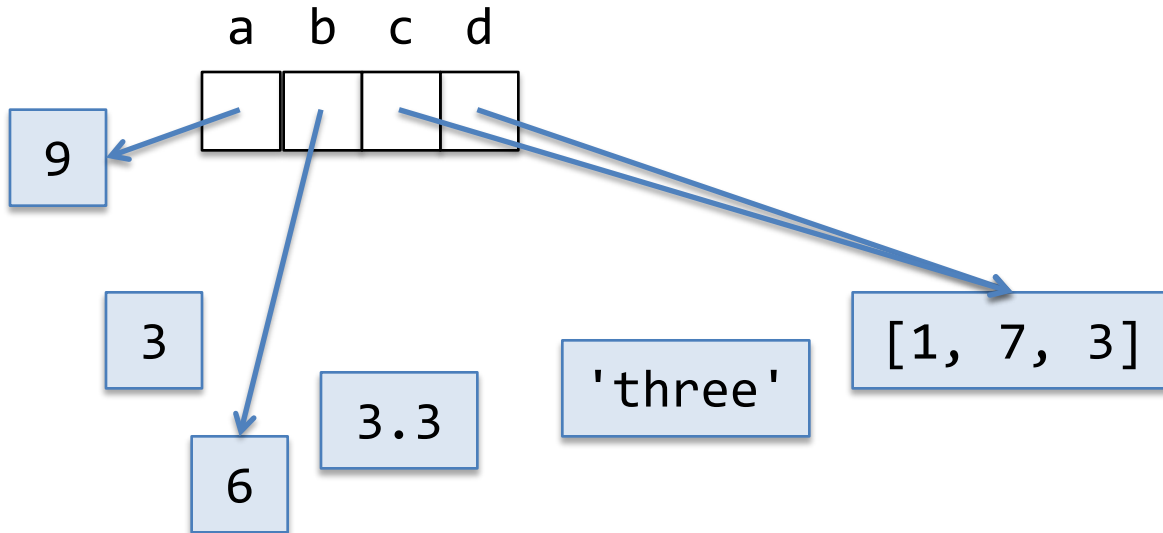
➤ **a**-ի փոփոխությունը չի փոխում

**b**-ի արժեքը

```
>>> a
6
>>> b
3.3
>>> b = a
>>> b
6
>>> a = 9
>>> b
6
```



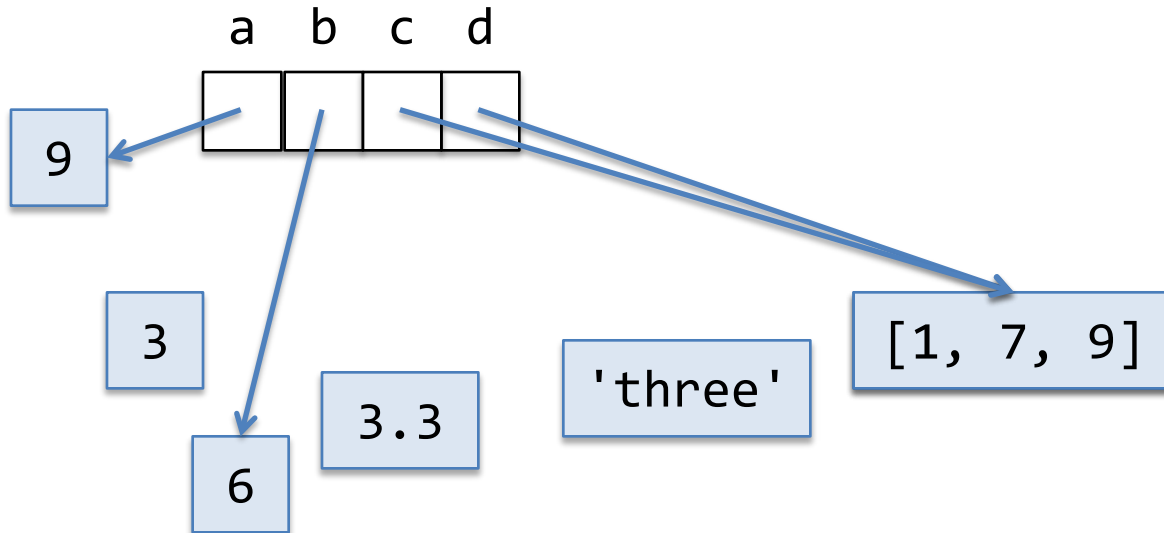
# Assignment and mutability



```
>>> a
6
>>> b
3.3
>>> b = a
>>> b
6
>>> a = 9
>>> b
6
>>> c = d
>>> c
[1, 7, 3]
```

`c` and `d` ցույց են տալիս նույն **list object**-ին

# Assignment and mutability



Երբ **c**-ի ցույց տրված object-ը փոխվում է;

**d**-ն նույնպես փոխվում է

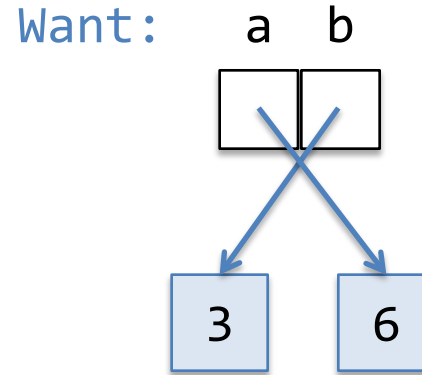
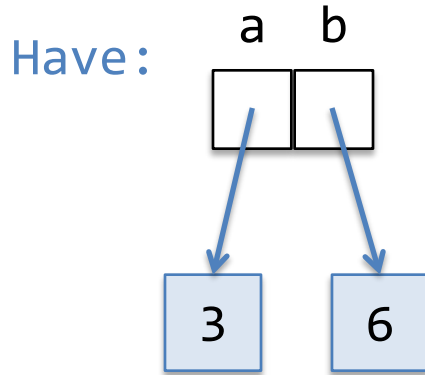
➤ lists → **mutable**

➤ **d**-ի փոփոխությունը փոխում է **c**

```
>>> a
6
>>> b
3.3
>>> b = a
>>> b
6
>>> a = 9
>>> b
6
>>> c = d
>>> c
[1, 7, 3]
>>> d[2] = 9
>>> c
[1, 7, 9]
```

# Swapping values

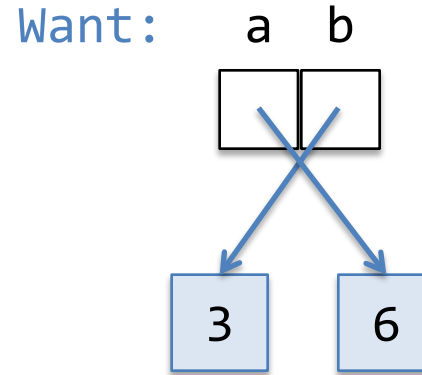
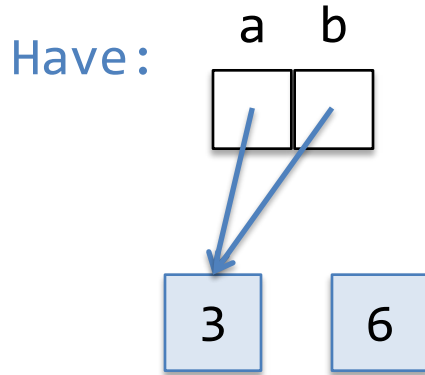
## Արժեքների փոխանակում



```
>>> a
3
>>> b
6
```

# Swapping values

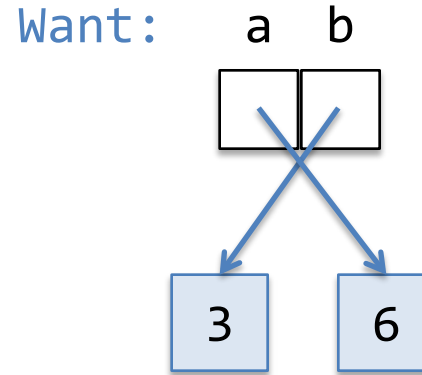
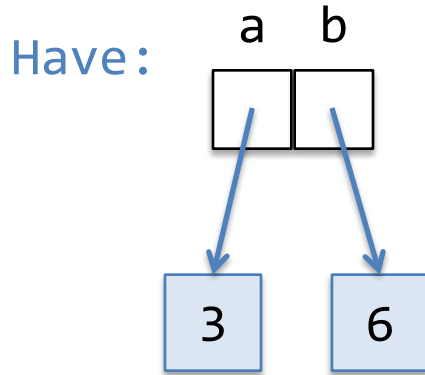
## Արժեքների փոխանակում



```
>>> a
3
>>> b
6
>>> b = a
```

# Swapping values

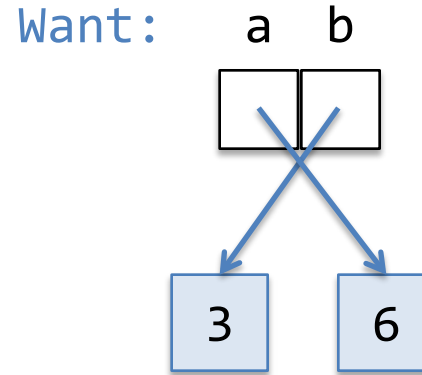
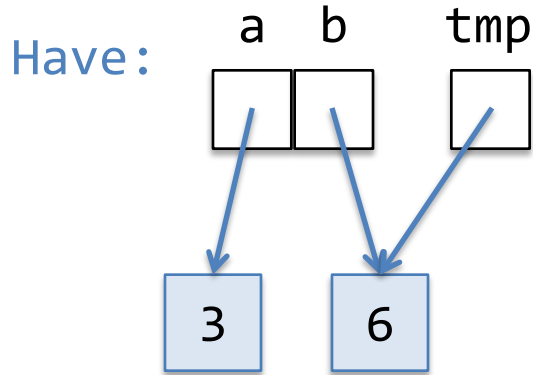
## Արժեքների փոխանակում



```
>>> a
3
>>> b
6
```

# Swapping values

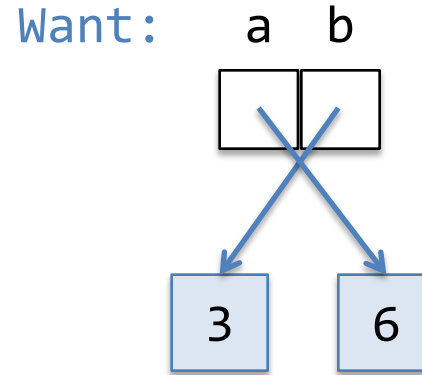
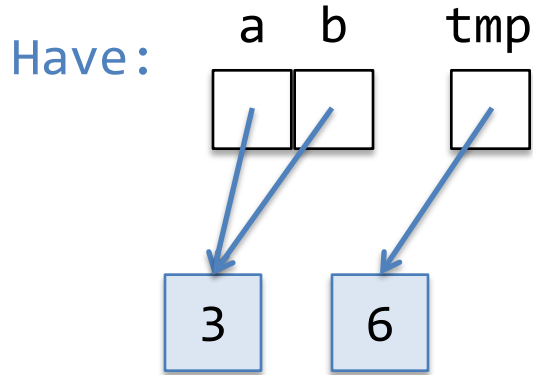
## Արժեքների փոխանակում



```
>>> a
3
>>> b
6
>>> tmp = b
```

# Swapping values

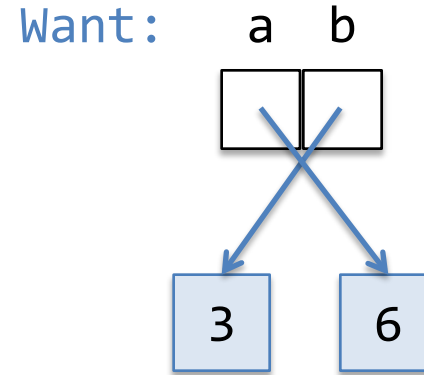
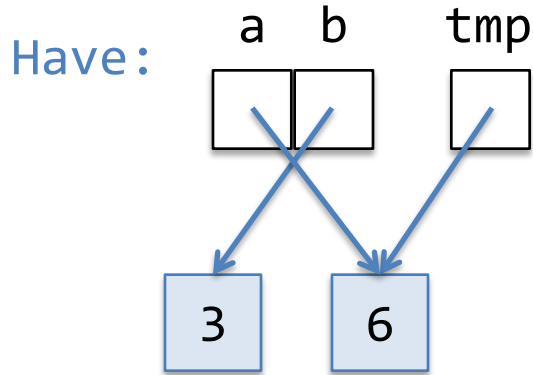
## Արժեքների փոխանակում



```
>>> a
3
>>> b
6
>>> tmp = b
>>> b = a
```

# Swapping values

## Արժեքների փոխանակում

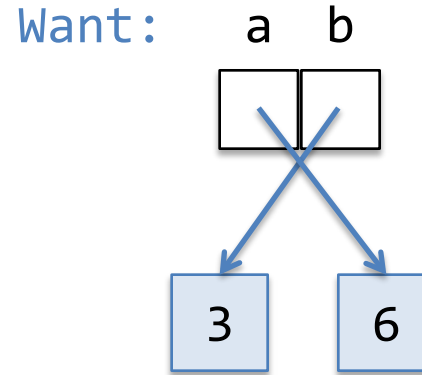
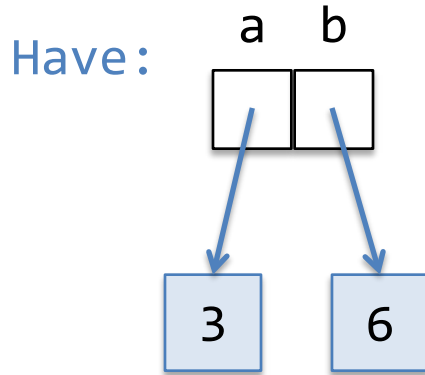


```
>>> a
3
>>> b
6
>>> tmp = b
>>> b = a
>>> a = tmp
```



# Swapping values

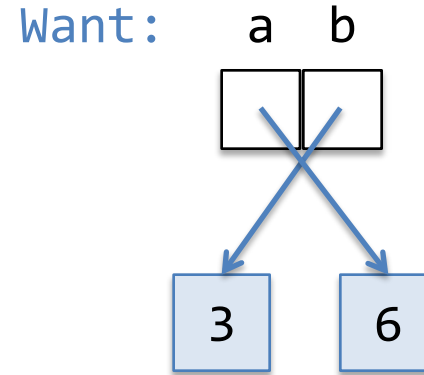
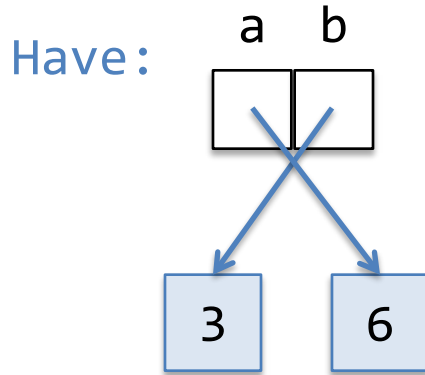
## Արժեքների փոխանակում



```
>>> a
3
>>> b
6
```

# Swapping values

## Արժեքների փոխանակում



```
>>> a
3
>>> b
6
>>> a, b = b, a
```

# Exercise 6A

Գրեք ծրագիր որը հաշվում է 1-ից 5 թվերի արտադրյալը

# Exercise 6A

Գրեք ծրագիր որը հաշվում է 1-ից 5 թվերի արտադրյալը

```
fact = 1
num = 5
for i in range(1,num+1):
    fact = fact*i
print(fact)
```

ex6.py

# Exercise 6

- Մահմանեք factorial function-ը
- Տպեք function-ի արժեքը 5 արժեքի համար

```
def factorial(num):  
    """ Calculate factorial """  
    # your code here  
    return fact  
  
a = factorial(5)  
print(a)
```

ex6.py

# Exercise 6

- Մահմանեք factorial function-ը
- Տպեք function-ի արժեքը 5 արժեքի համար

```
def factorial(num):  
    """ Calculate factorial """  
    fact = 1  
    for i in range(1,num+1):  
        fact = fact*i  
    return fact
```

```
a = factorial(5)  
print(a)
```

ex6.py

# pass statement

It does absolutely nothing.

```
def myfunc():  
    pass
```

```
for i in range(1000):  
    pass
```

# pass statement

It does absolutely nothing.

Programmers like to use it

- to waste time in some code or
- to hold the place to put real code at a later time.

```
def myfunc():  
    pass
```

```
for i in range(1000):  
    pass
```



# References

1. Franek. “CS 1MD3 Introduction to Programming.” Accessed July 8, 2014.
2. Downey, Allen B. *Think Python*. 1 edition. Sebastopol, CA: O’Reilly Media, 2012.
3. Guo, Philip. “Online Python Tutor - Visualize Program Execution.” *Pythontutor*. Accessed July 16, 2014. <http://pythontutor.com>.
4. Beckles, Bruce, and Bob Dowling. “Python: Introduction for Programmers — University Information Services (Academic & Infrastructure).” Accessed July 16, 2014.