



Introduction to Programming

Lesson 9

Outline

- Recursion
- Modules as namespaces
- Classes as namespaces

Recursion

recursive function – function որը
կանչում է ինքն իրեն

```
def countdown(n):  
    print(n)  
    countdown(n-1)
```

Recursion

recursive function – function որը

կանչում է ինքն իրեն

Ինչ կլինի եթե run տանք

countdown(3)?

```
def countdown(n):  
    print(n)  
    countdown(n-1)
```

Recursion

```
>>> countdown(3)
3
2
1
0
-1
-2
...
-976
-977
-978
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    countdown(3)
  File "/Users/me/ch10.py", line 3, in countdown
    countdown(n-1)
...
  File "/Users/me/ch10.py", line 3, in countdown
    countdown(n-1)
  File "/Users/me/ch10.py", line 2, in countdown
    print(n)
RuntimeError: maximum recursion depth exceeded
while calling a Python object
>>>
```

```
def countdown(n):
    print(n)
    countdown(n-1)
```

Recursion

```
>>> countdown(3)
3
2
1
0
-1
-2
...
-976
-977
-978
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    countdown(3)
  File "/Users/me/ch10.py", line 3, in countdown
    countdown(n-1)
...
  File "/Users/me/ch10.py", line 3, in countdown
    countdown(n-1)
  File "/Users/me/ch10.py", line 2, in countdown
    print(n)
RuntimeError: maximum recursion depth exceeded
while calling a Python object
>>>
```

```
def countdown(n):
    print(n)
    countdown(n-1)
```

function անընդհատ
կանչում է իրեն մինչև
resources are
exhausted/սպառվում են

- i.e., գերազանցվում է
program stack-ի limit-ը

Recursion

```
>>> countdown(3)
3
2
1
0
-1
-2
...
-976
-977
-978
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    countdown(3)
  File "/Users/me/ch10.py", line 3, in countdown
    countdown(n-1)
  ...
  File "/Users/me/ch10.py", line 3, in countdown
    countdown(n-1)
  File "/Users/me/ch10.py", line 2, in countdown
    print(n)
RuntimeError: maximum recursion depth exceeded
while calling a Python object
>>>
```

```
def countdown(n):
    print(n)
    countdown(n-1)
```

function անընդհատ
կանչում է իրեն մինչև
resources are
exhausted/սպառվում են

- i.e., գերազանցվում է
program stack-ի limit-ը

Որպեսզի function –ը
նորմալ terminate անի
պետք է լինի **stopping
condition/ դադարի
պայման**

Recursion

Ենթադրենք մեզ պետք է հետևյալը

```
>>> countdown(3)
3
2
1
Blastoff!!!
>>>
```

```
def countdown(n):
    print(n)
    countdown(n-1)
```

Որպեսզի function –ը
նորմալ terminate անի
պետք է լինի **stopping
condition/ դադարի
պայման**

Recursion

Ենթադրենք մեզ պետք է հետևյալը

```
>>> countdown(3)
3
2
1
Blastoff!!!
>>> countdown(1)
1
Blastoff!!!
>>>
```

```
def countdown(n):
    print(n)
    countdown(n-1)
```

Որպեսզի function –ը
նորմալ terminate անի
պետք է լինի **stopping
condition/ դադարի
պայման**

Recursion

Ենթադրենք մեզ պետք է հետևյալը

```
>>> countdown(3)
3
2
1
Blastoff!!!
>>> countdown(1)
1
Blastoff!!!
>>> countdown(0)
Blastoff!!!
>>>
```

```
def countdown(n):
    print(n)
    countdown(n-1)
```

Որպեսզի function –ը
նորմալ terminate անի
պետք է լինի **stopping
condition/ դադարի
պայման**

Recursion

Ենթադրենք մեզ պետք է հետևյալը

```
>>> countdown(3)
3
2
1
Blastoff!!!
>>> countdown(1)
1
Blastoff!!!
>>> countdown(0)
Blastoff!!!
>>> countdown(-1)
Blastoff!!!
```

```
def countdown(n):
    print(n)
    countdown(n-1)
```

Որպեսզի function –ը
նորմալ terminate անի
պետք է լինի **stopping
condition/ դադարի
պայման**

Recursion

Ենթադրենք մեզ պետք է հետևյալը


```
>>> countdown(3)
3
2
1
Blastoff!!!
>>> countdown(1)
1
Blastoff!!!
>>> countdown(0)
Blastoff!!!
>>> countdown(-1)
Blastoff!!!
```

If $n \leq 0$

'Blastoff!!!' is printed

```
def countdown(n):
    print(n)
    countdown(n-1)
```

Որպեսզի function-ը
նորմալ terminate անի
պետք է լինի **stopping
condition**/ դադարի
պայման



Recursion

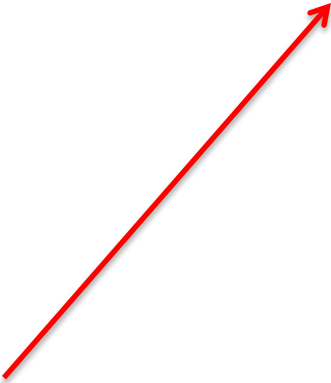
Ենթադրենք մեզ պետք է հետևյալը

```
>>> countdown(3)
3
2
1
Blastoff!!!
>>> countdown(1)
1
Blastoff!!!
>>> countdown(0)
Blastoff!!!
>>> countdown(-1)
Blastoff!!!
```

If $n \leq 0$

'Blastoff!!!' is printed

```
def countdown(n):
    'counts down to 0'
    if n <= 0:
        print('Blastoff!!!')
    else:
        print(n)
        countdown(n-1)
```



Որպեսզի function-ը
նորմալ terminate անի
պետք է լինի **stopping
condition/ դադարի
պայման**

Recursion

Base case →

Recursive step →

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Recursion

Base case →

Recursive step →

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Recursive function պետք է բաղկացած լինի

1. One or more **base cases** which provide the stopping condition of the recursion
Մեկ կամ մի քանի base case-երից որոնք կանգնեցնում են recursion-ը
2. One or more **recursive calls** on input arguments that are “closer” to the base case
Մեկ կամ մի քանի recursive call-երից, որոնց argument-ները ավելի մոտ են base case-ին

Recursion

Base case →

Recursive step →

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Recursive function պետք է բաղկացած լինի

1. One or more **base cases** which provide the stopping condition of the recursion
Մեկ կամ մի քանի base case-երից որոնք կանգնեցնում են recursion-ը
2. One or more **recursive calls** on input arguments that are “closer” to the base case
Մեկ կամ մի քանի recursive call-երից, որոնց argument-ները ավելի մոտ են base case-ին

Այս պայմանը ի վերջո կապահովի դադարի պայմանի իրականացումը

Recursive thinking

Recursive function-ը պետք է
բաղկացած լինի

1. Մեկ կամ մի քանի base case-երից
որոնք կանգնեցնում են recursion-ը
2. Մեկ կամ մի քանի recursive call-
երից, որոնց argument-ները ավելի
մոտ են base case-ին

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Recursive thinking

Recursive function-ը պետք է
բաղկացած լինի

1. Մեկ կամ մի քանի base case-երից
որոնք կանգնեցնում են recursion-ը
2. Մեկ կամ մի քանի recursive call-
երից, որոնց argument-ները ավելի
մոտ են base case-ին

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Հաշվել n-ից մինչև 0 ...

Recursive thinking

Recursive function-ը պետք է
բաղկացած լինի

1. Մեկ կամ մի քանի base case-երից
որոնք կանգնեցնում են recursion-ը
2. Մեկ կամ մի քանի recursive call-
երից, որոնց argument-ները ավելի
մոտ են base case-ին

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Հաշվել n -ից մինչև 0 ...

... տպում ենք n ու հաշվում $n-1$ -ից մինչև 0

Recursive thinking

Recursive function-ը պետք է
բաղկացած լինի

1. Մեկ կամ մի քանի base case-երից
որոնք կանգնեցնում են recursion-ը
2. Մեկ կամ մի քանի recursive call-
երից, որոնց argument-ները ավելի
մոտ են base case-ին

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Problem with input n

Հաշվել n-ից մինչև 0 ...

... տպում ենք n ու հաշվում n-1-ից մինչև 0

Recursive thinking

Recursive function-ը պետք է
բաղկացած լինի

1. Մեկ կամ մի քանի base case-երից
որոնք կանգնեցնում են recursion-ը
2. Մեկ կամ մի քանի recursive call-
երից, որոնց argument-ները ավելի
մոտ են base case-ին

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Problem with input n

Հաշվել n-ից մինչև 0 ...

... տպում ենք n ու հաշվում n-1-ից մինչև 0

Subproblem with input n-1

Recursive thinking

Recursive function-ը պետք է
բաղկացած լինի

1. Մեկ կամ մի քանի base case-երից
որոնք կանգնեցնում են recursion-ը
2. Մեկ կամ մի քանի recursive call-
երից, որոնց argument-ները ավելի
մոտ են base case-ին

```
def countdown(n):  
    'counts down to 0'  
    if n <= 0:  
        print('Blastoff!!!')  
    else:  
        print(n)  
        countdown(n-1)
```

Problem with input n

Հաշվել n-ից մինչև 0 ...

... տպում ենք n ու հաշվում n-1-ից մինչև 0

Subproblem with input n-1

=> Խնդիրը recursive լուծելու համար, մեզ պետք է

1. Սահմանել bases case/s որի համար problem-ը ունի լուծում
2. Արտահայտել problem-ի լուծումը **subproblem-ի լուծումի տեսքով**
(i.e., subproblem : easier instances of the problem that are closer to the bases cases)

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական `integer`-ներ և տպում ուղղահայացորեն

```
>>> vertical(3124)
```

```
3
```

```
1
```

```
2
```

```
4
```

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական `integer`-ներ և տպում ուղղահայացորեն

```
>>> vertical(3124)
3
1
2
4
```

Առաջինը սահմանենք `base case`-ը

- դեպք երբ `problem`-ը “հեշտ” է
- ո-ը մեկ հատ թիվ է

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական `integer`-ներ և տպում ուղղահայացորեն

```
>>> vertical(7)  
7
```

Առաջինը սահմանենք `base case`-ը

- դեպք երբ `problem`-ը “հեշտ” է
- ո-ը մեկ հատ թիվ է

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական `integer`-ներ և տպում ուղղահայացորեն

```
>>> vertical(7)
7
```

Առաջինը սահմանենք `base case`-ը

- դեպք երբ `problem`-ը “հեշտ” է
- `n`-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական integer-ներ և տպում ուղղահայացորեն

```
>>> vertical(7)
7
```

Տպիր `n`-ի թվերը ուղղահայաց...

Առաջինը սահմանենք `base case`-ը

- դեպք երբ `problem`-ը “հեշտ” է
- `n`-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական `integer`-ներ և տպում ուղղահայացորեն

```
>>> vertical(7)
7
```

Problem with input n

Տպիր n-ի թվերը ուղղահայաց...

Առաջինը սահմանենք `base case`-ը

- դեպք երբ `problem`-ը “հեշտ” է
- `n`-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական integer-ներ և տպում ուղղահայացորեն

```
>>> vertical(7)
7
```

Problem with input n

Տպիր n-ի թվերը ուղղահայաց...

... Տպիր n-ի բոլոր թվերը բացի վերջինից
և հետո տպիր վերջինը

Առաջինը սահմանենք `base case`-ը

- դեպք երբ problem-ը “հեշտ” է
- n-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական integer-ներ և տպում ուղղահայացորեն

```
>>> vertical(3124)
3
1
2
4
```

Problem with input n

Տպիր n-ի թվերը ուղղահայաց...

... Տպիր n-ի բոլոր թվերը բացի վերջինից
և հետո տպիր վերջինը

Առաջինը սահմանենք `base case`-ը

- դեպք երբ problem-ը “հեշտ” է
- n-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական integer-ներ և տպում ուղղահայացորեն

```
>>> vertical(3124)
3
1
2
4
```

Problem with input n

Տպիր n-ի թվերը ուղղահայաց...

... Տպիր n-ի բոլոր թվերը բացի վերջինից
և հետո տպիր վերջինը

Subproblem, որում n-ը մեկ թվով քիչ է

Առաջինը սահմանենք `base case`-ը

- դեպք երբ `problem`-ը “հեշտ” է
- n-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```

Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական integer-ներ և տպում ուղղահայացորեն

```
>>> vertical(3124)
3
1
2
4
```

Problem with input n

Տպիր n-ի թվերը ուղղահայաց...

... Տպիր n-ի բոլոր թվերը բացի վերջինից
և հետո տպիր վերջինը

Subproblem, որում n-ը մեկ թվով քիչ է

The last digit of n: $n \% 10$

Առաջինը սահմանենք `base case`-ը

- դեպք երբ problem-ը “հեշտ” է
- n-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```


Recursive thinking

Կառուցենք `function vertical()` որը վերցնում է դրական integer-ներ և տպում ուղղահայացորեն

```
>>> vertical(3124)
3
1
2
4
```

Առաջինը սահմանենք `base case`-ը

- դեպք երբ `problem`-ը “հեշտ” է
- `n`-ը մեկ հատ թիվ է

```
def vertical(n):
    '''prints digits of n
    vertically'''
    if n < 10:
        print(n)
    else:
        # to do
```

Problem with input `n`

Տպիր `n`-ի թվերը ուղղահայաց...

... Տպիր `n`-ի բոլոր թվերը բացի վերջինից
և հետո տպիր վերջինը

Subproblem, որում `n`-ը մեկ թվով քիչ է

The last digit of `n`: `n%10`

The integer obtained by removing the last digit of `n`: `n//10`

Recursive thinking

Հետո կառուցում ենք recursive step
Երբ n -ը ունի մեկից ավել թիվ

The last digit of n : $n \% 10$

The integer obtained by removing the last digit of n : $n // 10$

Recursive thinking

Հետո կառուցում ենք recursive step
Երբ n -ը ունի մեկից ավել թիվ

... Տպիր n -ի բոլոր թվերը
բացի վերջինից
և հետո տպիր վերջինը

The last digit of n : $n \% 10$

The integer obtained by removing the last digit of n : $n // 10$

Recursive thinking

Հետո կառուցում ենք recursive step
Երբ n -ը ունի մեկից ավել թիվ

... Տպիր n -ի բոլոր թվերը
բացի վերջինից
և հետո տպիր վերջինը

Subproblem, որում
 n -ը մեկ թվով քիչ է

The last digit of n : $n \% 10$

The integer obtained by removing the last digit of n : $n // 10$

Recursive thinking

Հետո կառուցում ենք recursive step
Երբ n -ը ունի մեկից ավել թիվ

... Տպիր n -ի բոլոր թվերը
բացի վերջինից
և հետո տպիր վերջինը

Subproblem, որում
 n -ը մեկ թվով քիչ է

```
def vertical(n):  
    '''prints digits of  
    n vertically'''  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

The last digit of n : $n\%10$

The integer obtained by removing the last digit of n : $n//10$

Exercise 1

Կառուցեք recursive function reverse()

1. որը վերցնում է դրական integer և տպում է իր թվերը ուղղահայաց
2. բայց տպում է սկսած վերջի թվից

```
>>> reverse(3124)
```

```
4
```

```
2
```

```
1
```

```
3
```

Exercise 1

Կառուցեք recursive function reverse()

1. որը վերցնում է դրական integer և տպում է իր թվերը ուղղահայաց
2. բայց տպում է սկսած վերջի թվից

```
>>> reverse(3124)
4
2
1
3
```

```
def reverse(n):
    '''prints digits of n vertically
    starting with low-order digit
    '''
    if n < 10:
        print(n)
    else:
        print(n%10)
        reverse(n//10)
```

Exercise 2

factorial function ունի հետևյալ recursive սահմանումը:

$$n! = n \times (n-1)! \quad \text{if } n > 0$$

$$0! = 1$$

Կառուցեք factorial() օգտագործելով recursion.

Exercise 2

factorial function ունի հետևյալ recursive սահմանումը:

$$n! = n \times (n-1)! \quad \text{if } n > 0$$
$$0! = 1$$

Կառուցեք factorial() օգտագործելով recursion.

```
def factorial(n):  
    '''returns the factorial  
    of integer n  
    '''  
    if n == 0:                # base case  
        return 1  
    return factorial(n-1)*n   # recursive step
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

```
n = 3124
```

```
vertical(3124)
```

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

```
n = 3124  
vertical(n//10)
```

```
vertical(3124)
```

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

```
n = 3124  
vertical(n//10)
```

```
vertical(3124)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

n = 312

vertical(312)

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

```
n = 312  
vertical(n//10)
```

```
vertical(312)
```


Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

```
n = 312  
vertical(n//10)
```

```
vertical(312)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

```
n = 31
```

```
vertical(31)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    '''  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

```
n = 31  
vertical(n//10)
```

```
vertical(31)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 31
line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    '''  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

```
n = 31  
vertical(n//10)
```

```
vertical(31)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 31
line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    '''  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(4312)
```

```
n = 3  
print(n)  
    vertical(3)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 31
line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    '''  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3
```

```
n = 3  
print(n)  
    vertical(3)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 31
line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    '''  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3
```

```
n = 31  
vertical(n//10)  
  
print(n%10)  
vertical(31)
```


Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    '''  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3  
1
```

```
n = 31  
vertical(n//10)  
  
print(n%10)  
vertical(31)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 312
line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3  
1
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3  
1
```

```
n = 312  
vertical(n//10)  
  
print(n%10)  
    vertical(312)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3  
1  
2
```

```
n = 312  
vertical(n//10)  
  
print(n%10)  
    vertical(312)
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

line = 7
n = 3124

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3  
1  
2
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

```
n = 3124  
vertical(n//10)
```

```
print(n%10)  
vertical(3124)
```

Program stack

```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3  
1  
2
```

Program stack

recursive function
calls հնարավոր է
դառնում program
stack-ի միջոցով
ինչպես և regular
function call-երը

```
n = 3124  
vertical(n//10)
```

```
print(n%10)  
vertical(3124)
```

Program stack

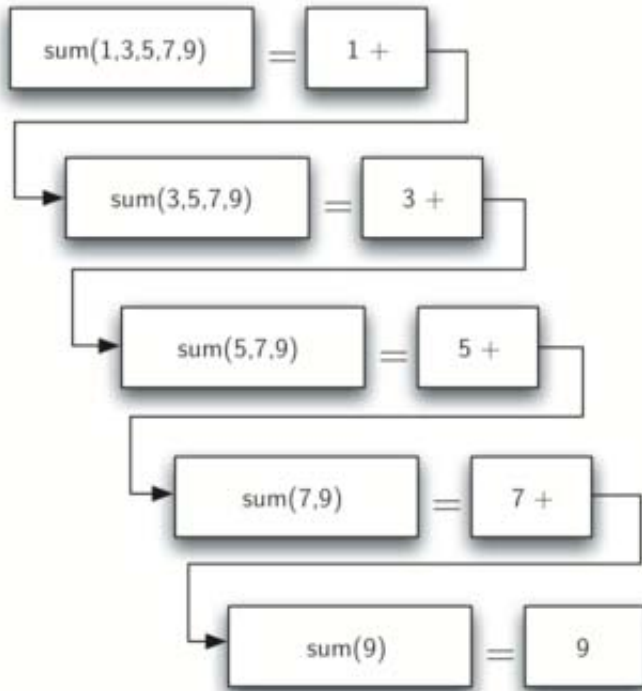
```
def vertical(n):  
    '''prints digits of  
    n vertically  
    ...  
    if n < 10:  
        print(n)  
    else:  
        vertical(n//10)  
        print(n%10)
```

```
>>> vertical(3124)  
3  
1  
2  
4
```

Program stack

```
def summa(numList):  
    if len(numList) == 1:  
        return numList[0]  
    else:  
        return numList[0] + summa(numList[1:])  
  
print(summa([1,3,5,7,9]))
```

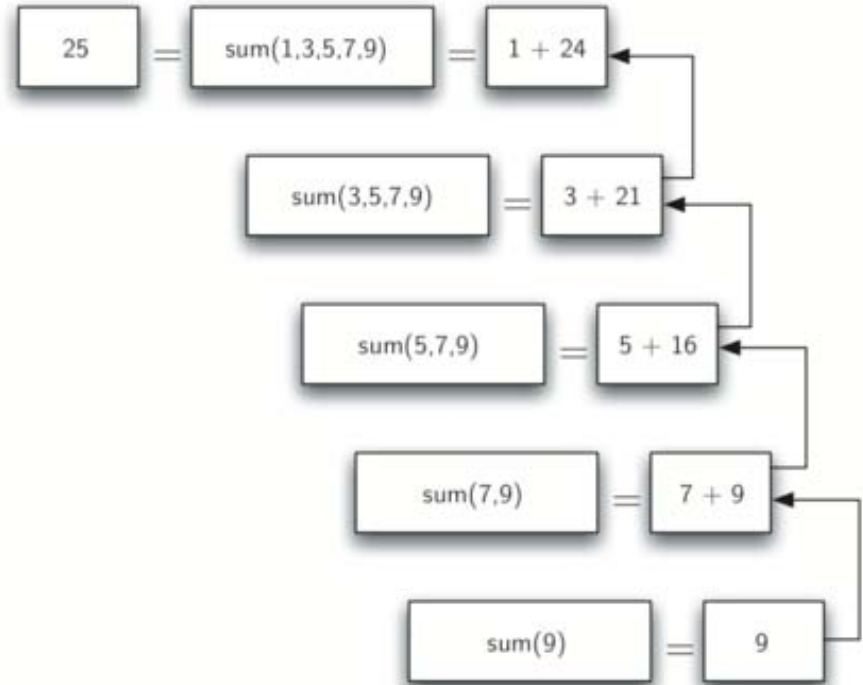
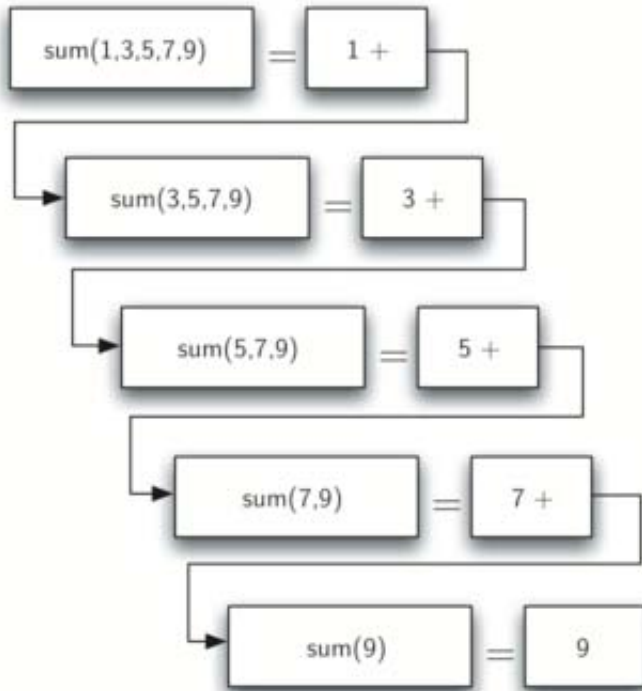

Program stack



```
def summa(numList):  
    if len(numList) == 1:  
        return numList[0]  
    else:  
        return numList[0] + summa(numList[1:])
```

```
print(summa([1,3,5,7,9]))
```

Program stack



```
def summa(numList):  
    if len(numList) == 1:  
        return numList[0]  
    else:  
        return numList[0] + summa(numList[1:])
```

```
print(summa([1,3,5,7,9]))
```

Exercise 3

Գրեք recursive function cheers()

1. որը վերցնում է integer input n,
2. և տպում է n հաստ string 'Hip '
որը հետևվում է 'Hurray!!! ' string-ով

```
>>> cheers(0)
Hurray!!!
>>> cheers(1)
Hip Hurray!!!
>>> cheers(4)
Hip Hip Hip Hip Hurray!!!
```

Exercise 3

Գրեք recursive function cheers()

1. որը վերցնում է integer input n,
2. և տպում է n անգամ string 'Hip '
որը հետևվում է 'Hurray!!! ' string-ով

```
>>> cheers(0)
Hurray!!!
>>> cheers(1)
Hip Hurray!!!
>>> cheers(4)
Hip Hip Hip Hip Hurray!!!
```

```
def cheers(n):
    if n == 0:
        print('Hurray!!!')
    else: # n > 0
        print('Hip', end=' ')
        cheers(n-1)
```

Modules

module-ը դա file է, որը պարունակում է Python code.

Երբ module-ը **import** էք անում, այն դառնում է **namespace**

Modules

module-ը դա file է, որը պարունակում է Python code.

Երբ module-ը **import** էք անում, այն դառնում է **namespace**

- այդ namespace-ը ունենում է նույն անունը (կարելի է փոխել)

Modules

module-ը դա file է, որը պարունակում է Python code.

Երբ module-ը **import** էք անում, այն դառնում է **namespace**

- այդ namespace-ը ունենում է նույն անունը (կարելի է փոխել)
- այդ namespace-ում ապրում են **names**, որոնք սահմանված են **module**-ի **global scope**-ում :
the names of functions, values, and classes

Modules

module-ը դա file է, որը պարունակում է Python code.

Երբ module-ը **import** էք անում, այն դառնում է **namespace**

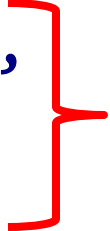
- այդ namespace-ը ունենում է նույն անունը (կարելի է փոխել)
- այդ namespace-ում ապրում են **names**, որոնք սահմանված են **module**-ի **global scope**-ում :
the names of functions, values, and classes
- these **names** are called **attributes** of the module

Modules

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__',
'acos', ..., 'pi', 'pow', 'radians',
'sin', 'sinh', 'sqrt', 'tan', 'tanh']
>>> math.sqrt
<built-in function sqrt>
>>> math.pi
3.141592653589793
```

Modules

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__',
'acos', ..., 'pi', 'pow', 'radians',
'sin', 'sinh', 'sqrt', 'tan', 'tanh']
>>> math.sqrt
<built-in function sqrt>
>>> math.pi
3.141592653589793
```



attributes

importing modules

import statement : միայն վերցնում է module-ի անունը

- without any directory information or .py suffix.

module-ը գտնելու համար Python օգտագործում է Python search path-ը.

- search path : list of directories որտեղ Python-ը փնտրում է module-ը
- sys.path : պարունակում է այդ list of directories

importing modules

import statement : միայն վերցնում է module-ի անունը

- without any directory information or .py suffix.

module-ը գտնելու համար Python օգտագործում է Python search path-ը.

- search path : list of directories որտեղ Python-ը փնտրում է module-ը
- sys.path : պարունակում է այդ list of directories

```
>>> import sys
```

importing modules

import statement : միայն վերցնում է module-ի անունը

- without any directory information or .py suffix.

module-ը գտնելու համար Python օգտագործում է Python search path-ը.

- search path : list of directories որտեղ Python-ը փնտրում է module-ը
- sys.path : պարունակում է այդ list of directories

```
>>> import sys
```

```
>>> sys.path
```

```
['', 'C:\\Python34\\Lib\\idlelib',  
'C:\\Python34\\python34.zip', 'C:\\Python34\\DLLs',  
'C:\\Python34\\lib', 'C:\\Python34',  
'C:\\Python34\\lib\\site-packages']
```

importing modules

import statement : միայն վերցնում է module-ի անունը

- without any directory information or .py suffix.

module-ը գտնելու համար Python օգտագործում է Python search path-ը.

- search path : list of directories որտեղ Python-ը փնտրում է module-ը
- sys.path : պարունակում է այդ list of directories

```
>>> import sys
```

```
>>> sys.path
```

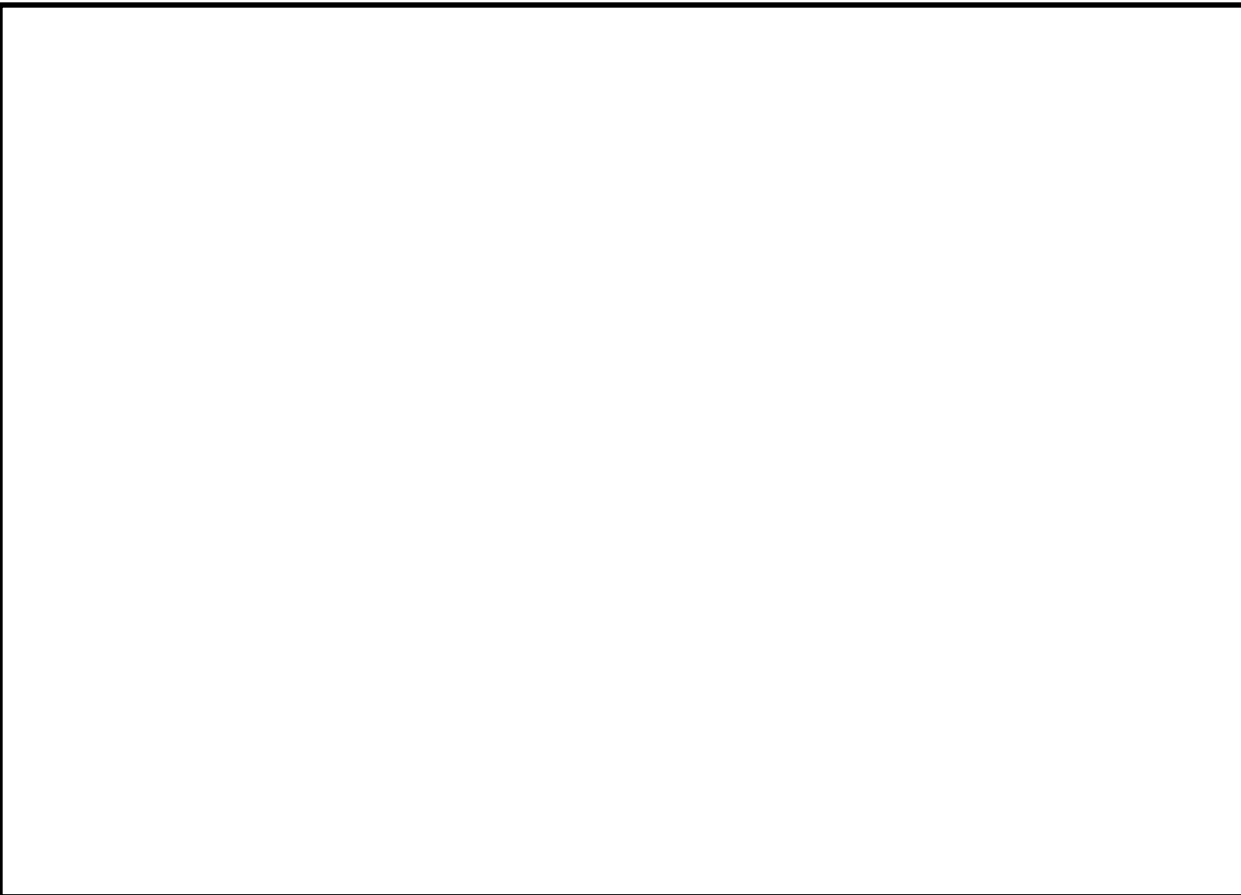
```
['', 'C:\\Python34\\Lib\\idlelib',  
'C:\\Python34\\python34.zip', 'C:\\Python34\\DLLs',  
'C:\\Python34\\lib', 'C:\\Python34',  
'C:\\Python34\\lib\\site-packages']
```

Standard Library folder



Python search path

Պետք է import անել module example, որը Desktop-ին է.
Desktop-ի directory-ն sys.path-ում չի



Python search path

Պետք է import անել module example, որը Desktop-ին է.
Desktop-ի directory-ն sys.path-ում չի

```
'an example module'  
def f():  
    'function f'  
    print('Executing f()')  
  
def g():  
    'function g'  
    print('Executing g()')  
  
x = 0 # global var
```

example.py

Python search path

Պենք է import անել module example, որը Desktop-ին է.
Desktop-ի directory-ն sys.path-ում չի

```
>>> import example
...
ImportError: No module named 'example'
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
```

example.py

Python search path

Պէտք է import անել module example, որը Desktop-ին է.
Desktop-ի directory-ն sys.path-ում չի

```
>>> import example
...
ImportError: No module named 'example'
>>> import sys
>>> sys.path.append(r'c:\Users\training\Desktop')
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
```

example.py

Python search path

Պէտք է import անել module example, որը Desktop-ին է.
Desktop-ի directory-ն sys.path-ում չի

```
>>> import example
...
ImportError: No module named 'example'
>>> import sys
>>>
sys.path.append(r'c:\Users\training\Desktop')
>>> import example
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
```

example.py

Python search path

Պէտք է import անել module example, որը Desktop-ին է.
Desktop-ի directory-ն sys.path-ում չի

```
>>> import example
...
ImportError: No module named 'example'
>>> import sys
>>>
sys.path.append(r'c:\Users\training\Desktop')
>>> import example
>>> example.f
<function f at 0x02316B70>
>>> example.x
0
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
```

example.py

Python search path

Պէտք է import անել module example, որը Desktop-ին է.
Desktop-ի directory-ն sys.path-ում չի

```
>>> import example
...
ImportError: No module named 'example'
>>> import sys
>>>
sys.path.append(r'c:\Users\training\Desktop')
>>> import example
>>> example.f
<function f at 0x02316B70>
>>> example.x
0
>>> dir()
['__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__', 'cls',
 'd', 'example', 'f', 'l', 'sys']
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
```

example.py

Top level module

Երբ module-ը import է արվում, Python-ը ստեղծում է մի քանի “bookkeeping” variable-ներ module-ի namespace-ում. Նրանցից մեկը `__name__` variable-ն է

Top level module

Երբ module-ը import է արվում, Python-ը ստեղծում է մի քանի “bookkeeping” variable-ներ module-ի namespace-ում. Նրանցից մեկը `__name__` variable-ն է

- `__name__ == '__main__'`, եթե module-ը վազում է որպես **top-level** module

Top level module

Երբ module-ը import է արվում, Python-ը ստեղծում է մի քանի “bookkeeping” variable-ներ module-ի namespace-ում. Նրանցից մեկը `__name__` variable-ն է

- `__name__ == '__main__'`, եթե module-ը վազում է որպես **top-level** module
- `__name__ == module name`, եթե import է արվել

Top level module

Երբ module-ը import է արվում, Python-ը ստեղծում է մի քանի “bookkeeping” variable-ներ module-ի namespace-ում. Նրանցից մեկը `__name__` variable-ն է

- `__name__ == '__main__'`, եթե module-ը վազում է որպես **top-level module**
- `__name__ == module name`, եթե import է արվել

A module is a **top-level module** if:

- it is **run from the shell (F5)**
- it is **run at the command line (cmd, terminal)**

Top level module

Երբ module-ը import է արվում, Python-ը ստեղծում է մի քանի “bookkeeping” variable-ներ module-ի namespace-ում. Նրանցից մեկը `__name__` variable-ն է

- `__name__ == '__main__'`, եթե module-ը վազում է որպես **top-level** module
- `__name__ == module name`, եթե import է արվել

A module is a **top-level module** if:

- it is **run from the shell (F5)**
- it is **run at the command line (cmd, terminal)**

```
print('My name is {}'.format(__name__))  
name.py
```

Top level module

Երբ module-ը import է արվում, Python-ը ստեղծում է մի քանի “bookkeeping” variable-ներ module-ի namespace-ում. Նրանցից մեկը `__name__` variable-ն է

- `__name__ == '__main__'`, եթե module-ը վազում է որպես **top-level** module
- `__name__ == module name`, եթե import է արվել

A module is a **top-level module** if:

- it is **run from the shell (F5)**
- it is **run at the command line (cmd, terminal)**

```
> python name.py
My name is __main__
```

```
print('My name is {}'.format(__name__))
name.py
```

Top level module

Երբ module-ը import է արվում, Python-ը ստեղծում է մի քանի “bookkeeping” variable-ներ module-ի namespace-ում. Նրանցից մեկը `__name__` variable-ն է

- `__name__ == '__main__'`, եթե module-ը վազում է որպես **top-level** module
- `__name__ == module name`, եթե import է արվել

A module is a **top-level module** if:

- it is **run from the shell (F5)**
- it is **run at the command line (cmd, terminal)**

```
>>>
My name is __main__
```

```
print('My name is {}'.format(__name__))
name.py
```

Top level module

```
%cd C:\Users\Generosity\Downloads\WinPython-32bit-3.3.5.0\data
```

```
C:\Users\Generosity\Downloads\WinPython-32bit-3.3.5.0\data
```

```
%%writefile name.py
```

```
print('My name is {}'.format(__name__))
```

```
Writing name.py
```

```
!python name.py
```

```
My name is __main__
```

```
%run name.py
```

```
My name is __main__
```

```
import name
```

```
My name is name
```

3 ways to import a module

1. Import the (name of the) module

```
>>>
```

```
namespace __main__
```

```
'an example module'  
def f():  
    'function f'  
    print('Executing f()')  
  
def g():  
    'function g'  
    print('Executing g()')  
  
x = 0 # global var
```

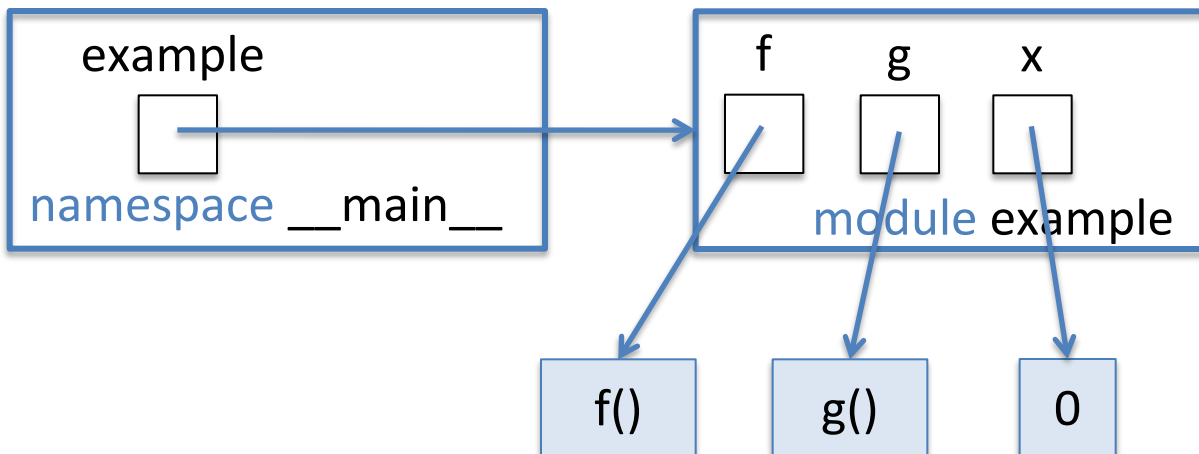
example.py

3 ways to import a module

1. Import the (name of the) module

```
>>> import example  
>>>
```

```
'an example module'  
def f():  
    'function f'  
    print('Executing f()')  
  
def g():  
    'function g'  
    print('Executing g()')  
  
x = 0 # global var  
example.py
```



3 ways to import a module

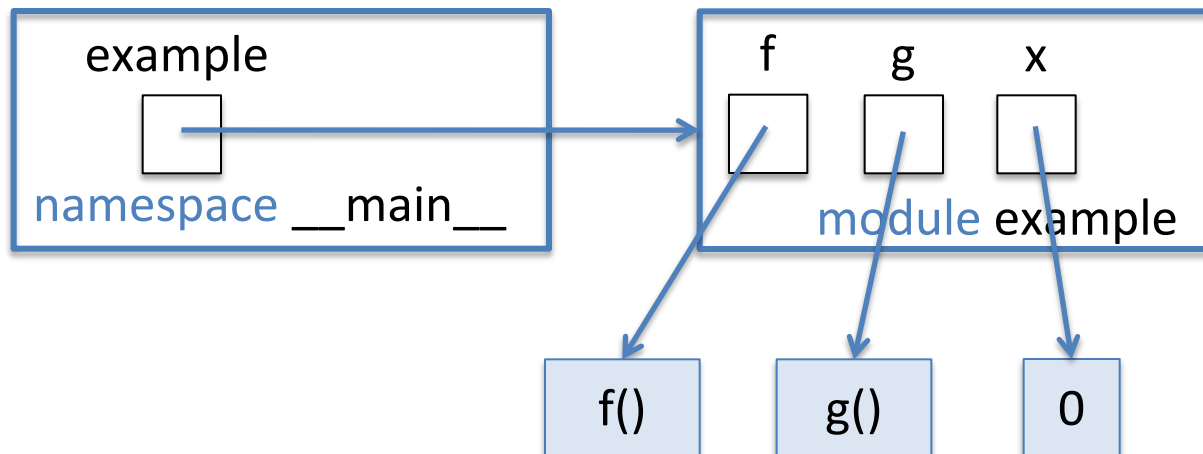
1. Import the (name of the) module

```
>>> import example
>>> example.x
0
>>> example.f
<function f at 0x10278dd98>
>>> example.f()
Executing f()
>>>
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
example.py
```



importing module attributes

2. Import specific module attributes

```
>>>
```

```
'an example module'  
def f():  
    'function f'  
    print('Executing f()')  
  
def g():  
    'function g'  
    print('Executing g()')  
  
x = 0 # global var
```

example.py

```
namespace __main__
```

importing module attributes

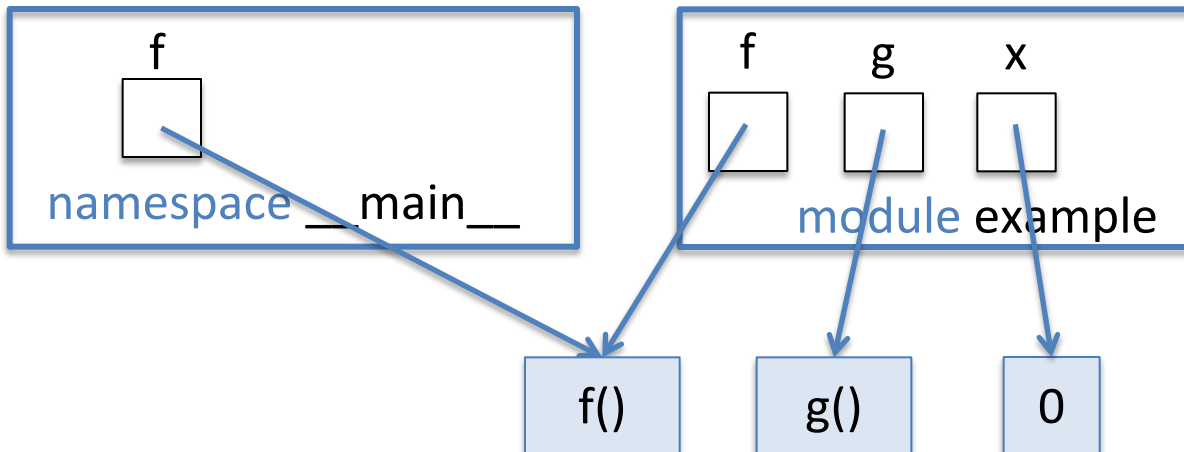
2. Import specific module attributes

```
>>> from example import f
>>>
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
example.py
```



importing module attributes

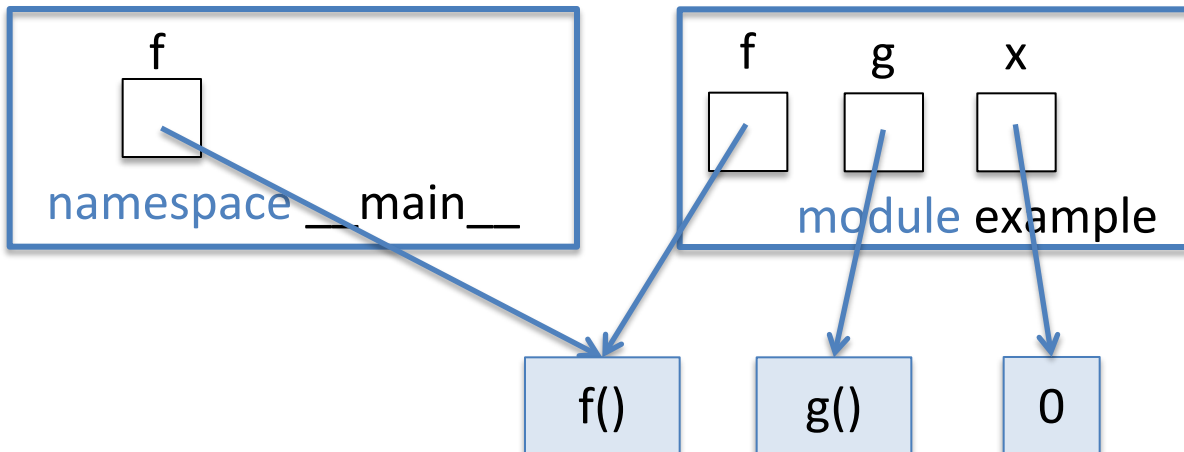
2. Import specific module attributes

```
>>> from example import f
>>> f()
Executing f()
>>> x
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    x
NameError: name 'x' is not defined
>>>
```

```
'an example module'
def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

x = 0 # global var
example.py
```



importing all attributes

3. Import all module attributes

```
>>>
```

```
'an example module'  
def f():  
    'function f'  
    print('Executing f()')  
  
def g():  
    'function g'  
    print('Executing g()')  
  
x = 0 # global var
```

example.py

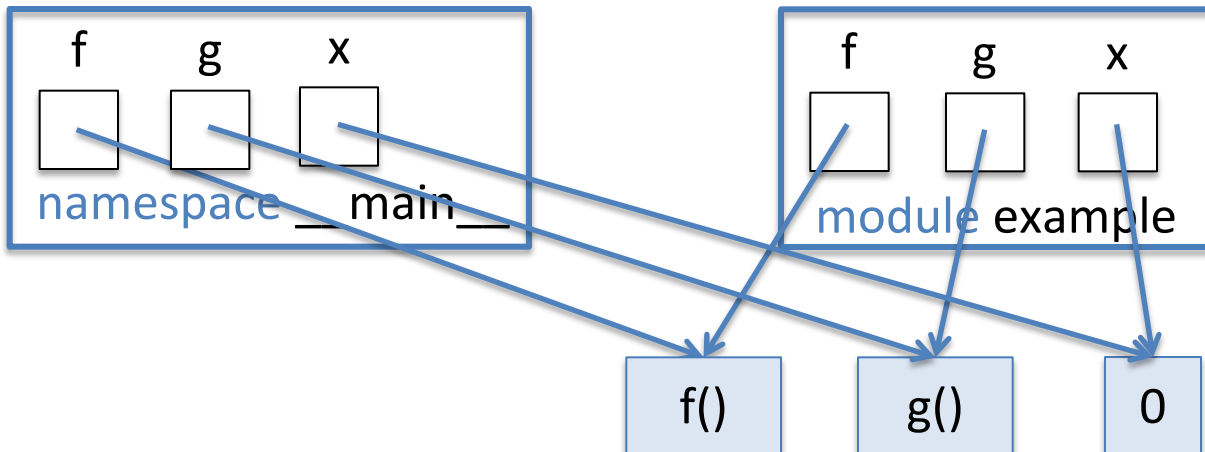
```
namespace __main__
```

importing all attributes

3. Import all module attributes

```
>>> from example import *  
>>>
```

```
'an example module'  
def f():  
    'function f'  
    print('Executing f()')  
  
def g():  
    'function g'  
    print('Executing g()')  
  
x = 0 # global var  
example.py
```

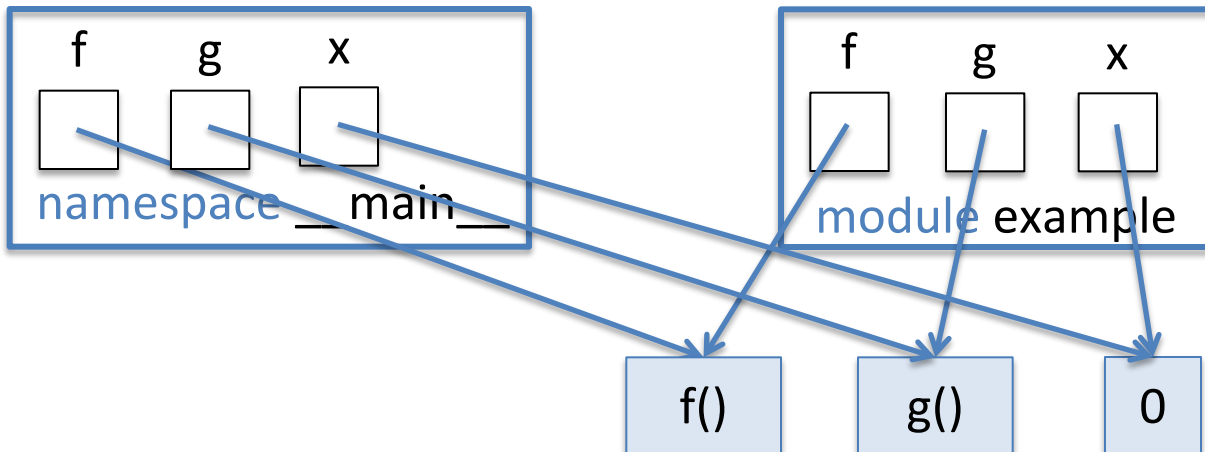


importing all attributes

3. Import all module attributes

```
>>> from example import *  
>>> f()  
Executing f()  
>>> g()  
Executing g()  
>>> x  
0  
>>>
```

```
'an example module'  
def f():  
    'function f'  
    print('Executing f()')  
  
def g():  
    'function g'  
    print('Executing g()')  
  
x = 0 # global var  
example.py
```



Exercise 4

Կառուցել module

```
import <>
```

```
def <>:  
    pass
```

```
def <>:  
    pass
```

```
def main():  
    pass
```

```
if __name__ == "__main__":  
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները

```
import <>
```

```
def <>:  
    pass
```

```
def <>:  
    pass
```

```
def main():  
    pass
```

```
if __name__ == "__main__":  
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները
- module-ը պետք է ունենա main function

```
import <>
```

```
def <>:  
    pass
```

```
def <>:  
    pass
```

```
def main():  
    pass
```

```
if __name__ == "__main__":  
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները
- module-ը պետք է ունենա main function
- նաև `if __name__ == "__main__":` condition

```
import <>
```

```
def <>:  
    pass
```

```
def <>:  
    pass
```

```
def main():  
    pass
```

```
if __name__ == "__main__":  
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները

- module-ը պետք է ունենա main function

- նաև `if __name__ == "__main__"` condition

- տպի իր անունը

```
print('My name is{} '
      .format(__name__))
```

```
import <>
```

```
def <>:
    pass
```

```
def <>:
    pass
```

```
def main():
    pass
```

```
if __name__ == "__main__":
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները

- module-ը պետք է ունենա main function

- նաև `if __name__ == "__main__"` condition

- տպի իր անունը

```
print('My name is{} '
      .format(__name__))
```

- կանչի module-ի որևէ function main-ից

```
import <>
```

```
def <>:
    pass
```

```
def <>:
    pass
```

```
def main():
    pass
```

```
if __name__ == "__main__":
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները

- module-ը պետք է ունենա main function

- նաև `if __name__ == "__main__"` condition

- տպի իր անունը

```
print('My name is{} '
      .format(__name__))
```

- կանչի module-ի որևէ function main-ից

- տպի `__doc__` variable-ը

```
import <>
```

```
def <>:  
    pass
```

```
def <>:  
    pass
```

```
def main():  
    pass
```

```
if __name__ == "__main__":  
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները

- module-ը պետք է ունենա main function

- նաև `if __name__ == "__main__"` condition

- տպի իր անունը

```
print('My name is{} '
      .format(__name__))
```

- կանչի module-ի նրկե function main-ից

- տպի `__doc__` variable-ը

- տպի `__file__` variable-ը

```
import <>
```

```
def <>:
    pass
```

```
def <>:
    pass
```

```
def main():
    pass
```

```
if __name__ == "__main__":
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները

- module-ը պետք է ունենա main function

- նաև `if __name__ == "__main__"` condition

- տպի իր անունը

```
print('My name is{ } '.  
      .format(__name__))
```

- կանչի module-ի որևէ function main-ից

- տպի `__doc__` variable-ը

- տպի `__file__` variable-ը

- ունենա global variable-ներ

```
import <>
```

```
def <>:  
    pass
```

```
def <>:  
    pass
```

```
def main():  
    pass
```

```
if __name__ == "__main__":  
    main()
```

example.py

Exercise 4

Կառուցեք module

- օգտագործեք ձեր գրած function-ները

- module-ը պետք է ունենա main function

- նաև `if __name__ == "__main__"` condition

- տպի իր անունը

```
print('My name is{} '.format(__name__))
```

- կանչի module-ի որևէ function main-ից

- տպի `__doc__` variable-ը

- տպի `__file__` variable-ը

- ունենա global variable-ներ

```
import <>
```

```
def <>:  
    pass
```

```
def <>:  
    pass
```

```
def main():  
    pass
```

```
if __name__ == "__main__":  
    main()
```

example.py

Ստուգեք module-ը:

- **shell-ում (F5)**
- **command line-ից (cmd/terminal)**
- **import արեք module-ը shell**

Solution 4

```
'an example module'

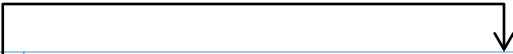
from math import sqrt, pow

x = 7 # global var
y = ['Hayer!']

def f():
    'function f'
    print('Executing f()')

def g():
    'function g'
    print('Executing g()')

def h(a,b):
    return sqrt(pow(a,2)+pow(b,2))
```



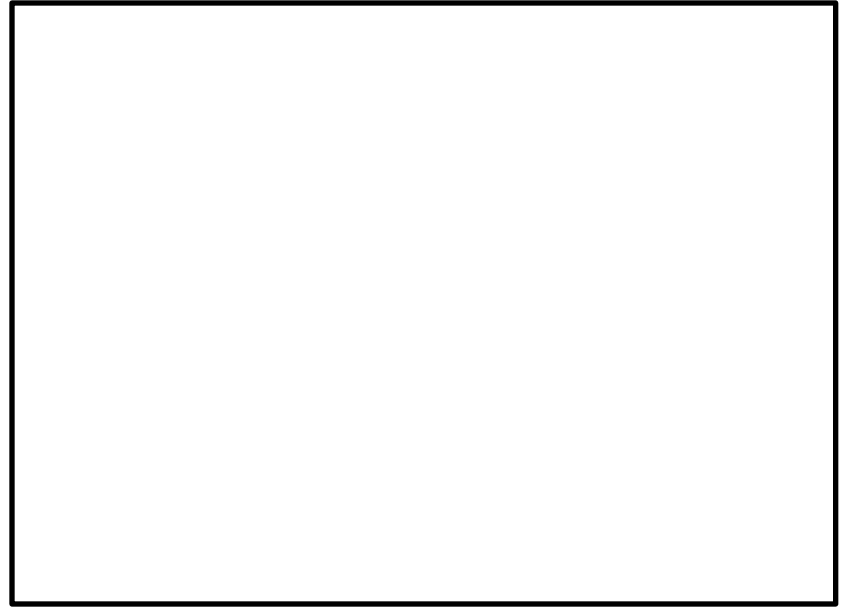
```
def main():
    print('My name is{} '
          .format(__name__))
    print('My documentation:{} '
          .format(__doc__))
    print('file directory:{} '
          .format(__file__))
    c = h(a=3,b=4)
    print('c =',c)

if __name__ == "__main__":
    main()
```

example.py

A class is a namespace

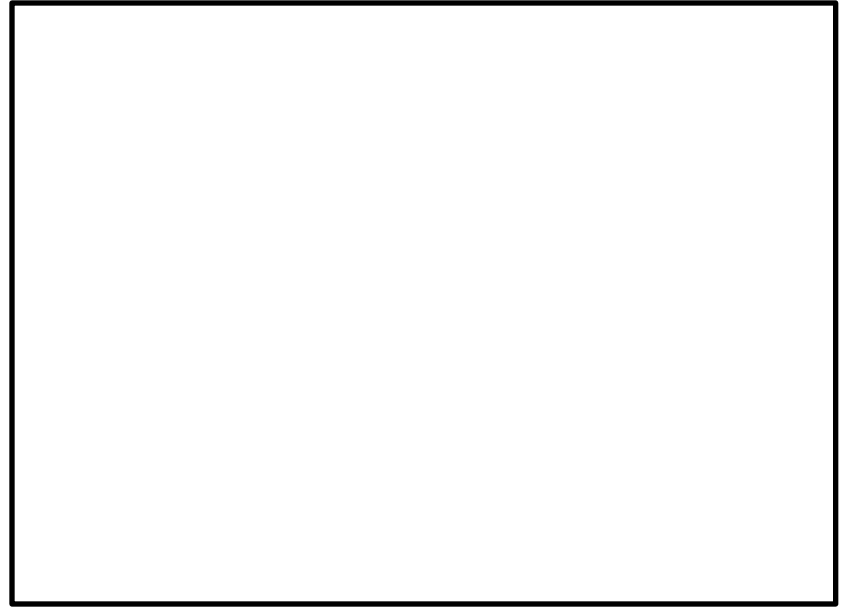
A class is really a namespace



A class is a namespace

A class is really a namespace

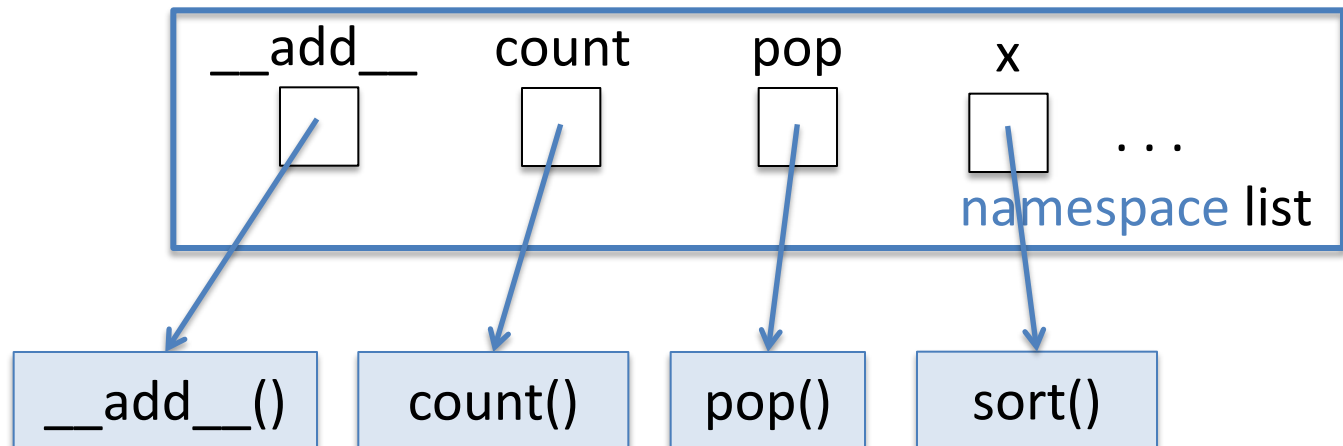
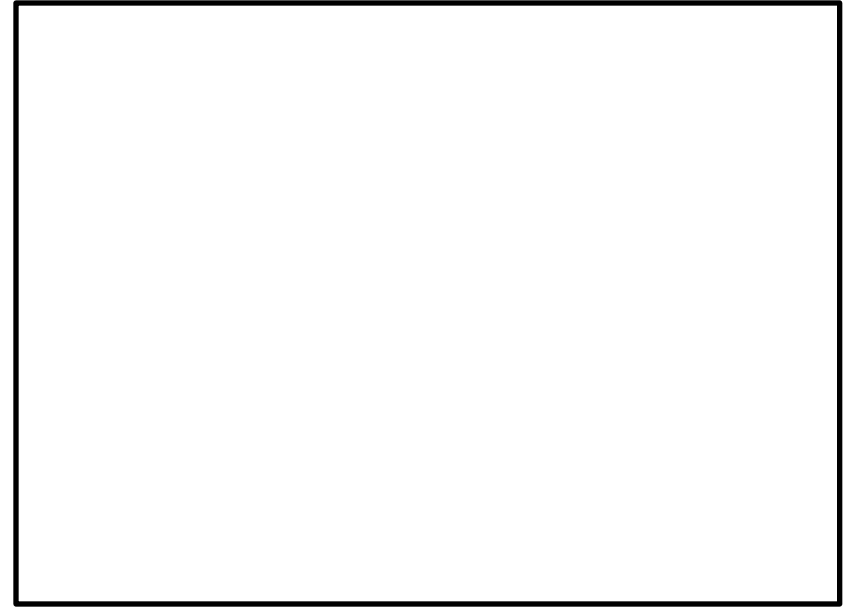
- The name of the namespace
== name of the class



A class is a namespace

A class is really a namespace

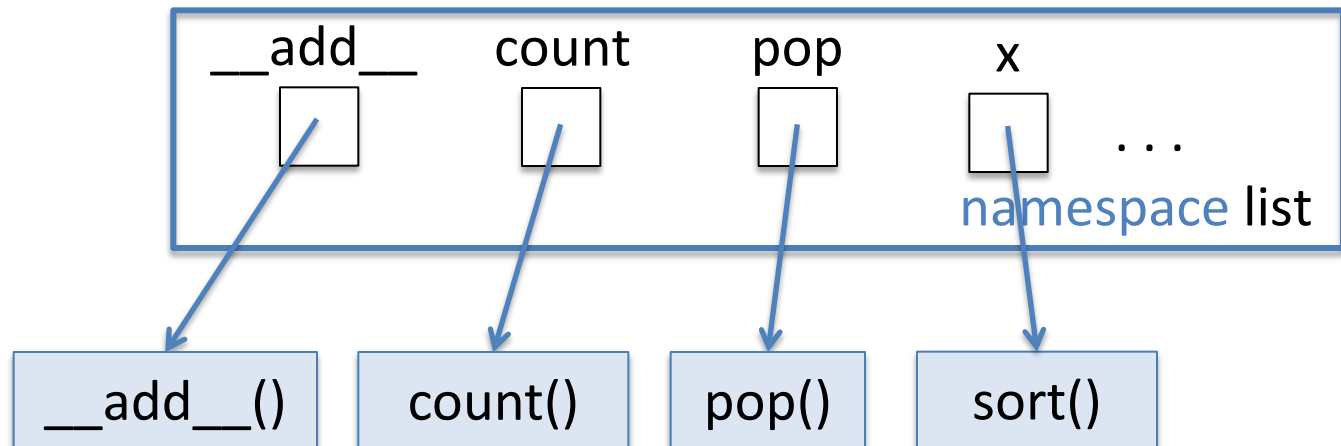
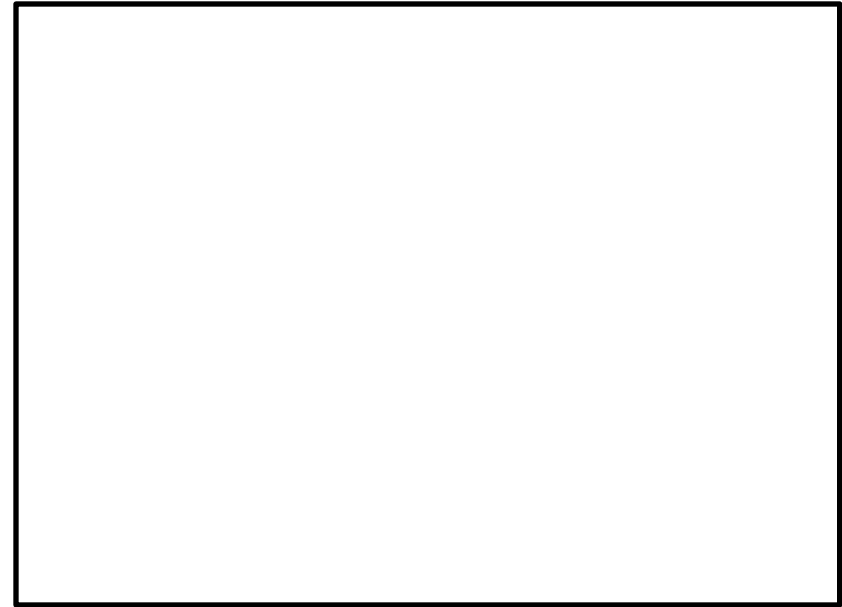
- The name of the namespace
== name of the class



A class is a namespace

A class is really a namespace

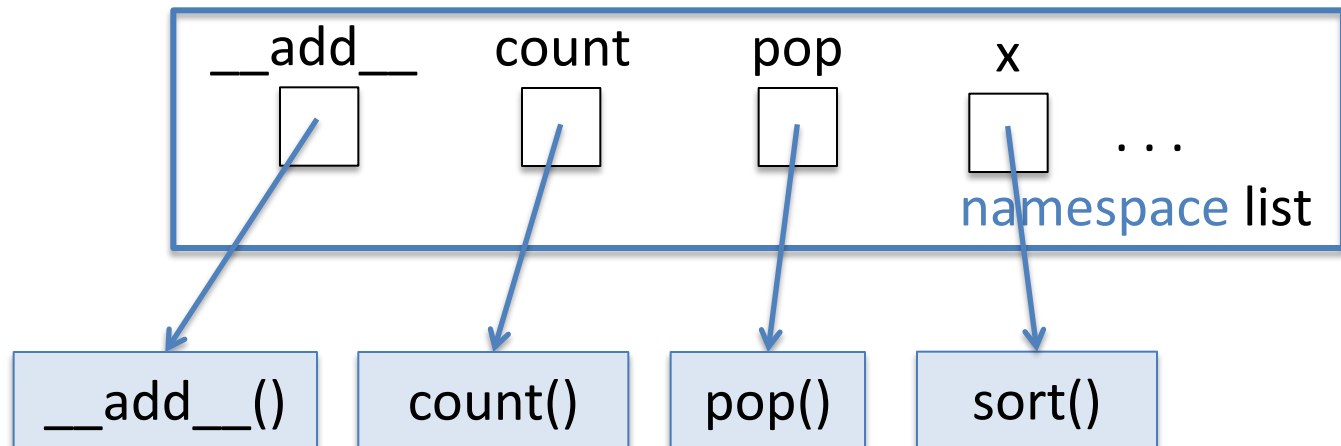
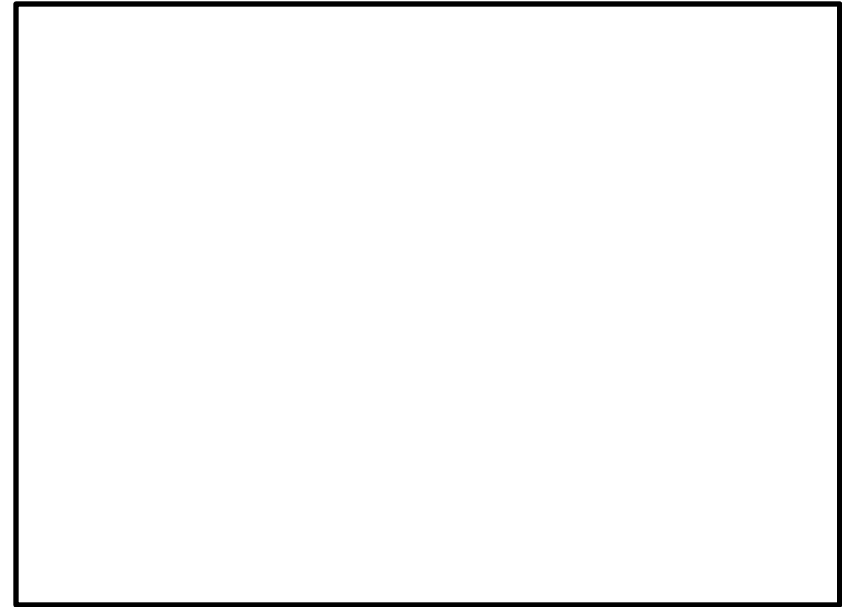
- The name of the namespace == name of the class
- names in this namespace are the class attributes (e.g., class methods)



A class is a namespace

A class is really a namespace

- The name of the namespace == name of the class
- names in this namespace are the class attributes (e.g., class methods)
- The class attributes accessed using the standard namespace notation '.' operator

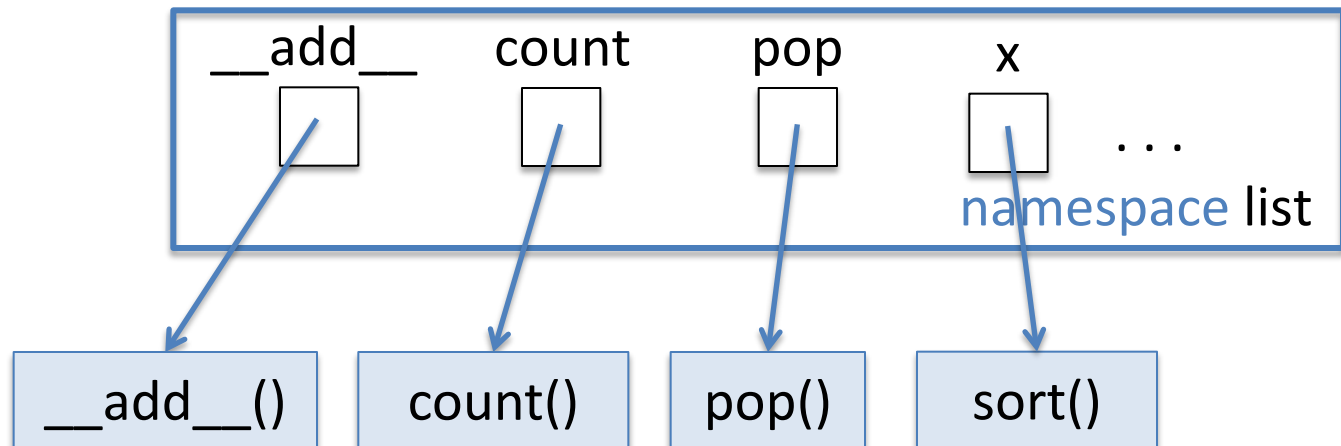


A class is a namespace

A class is really a namespace

- The name of the namespace == name of the class
- names in this namespace are the class attributes (e.g., class methods)
- The class attributes accessed using the standard namespace notation '.' operator

```
>>> list.pop  
<method 'pop' of 'list' objects>  
>>> list.sort  
<method 'sort' of 'list' objects>
```

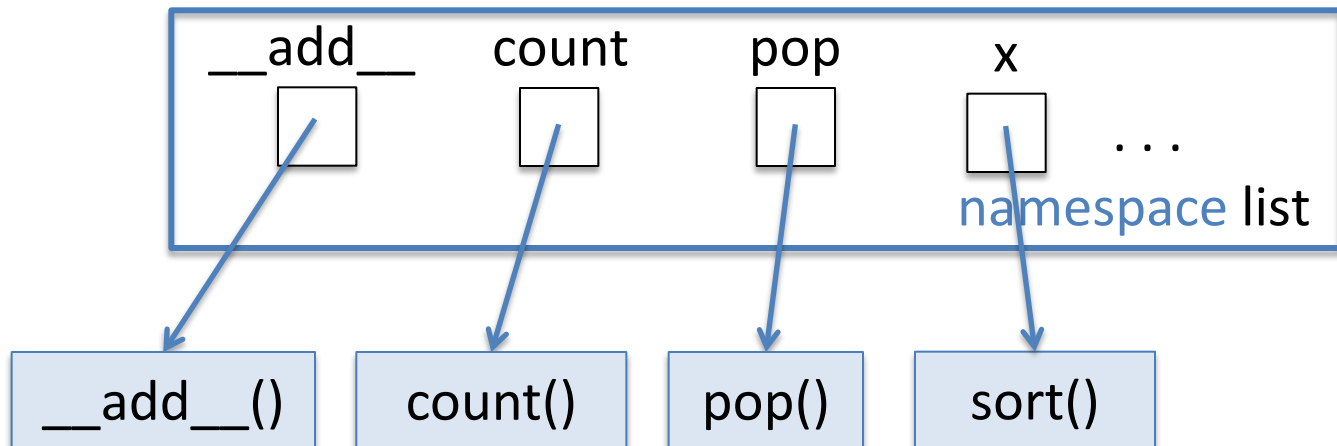


A class is a namespace

A class is really a namespace

- The name of the namespace == name of the class
- names in this namespace are the class attributes (e.g., class methods)
- The class attributes accessed using the standard namespace notation '.' operator

```
>>> list.pop
<method 'pop' of 'list' objects>
>>> list.sort
<method 'sort' of 'list' objects>
>>> dir(list)
['__add__', '__class__',
...
'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```



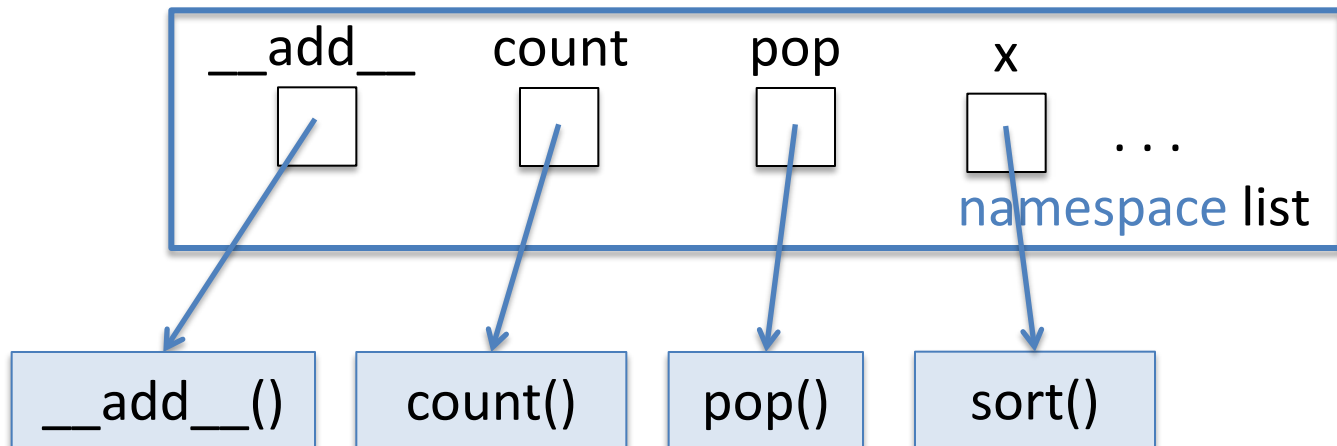
A class is a namespace

A class is really a namespace

- The name of the namespace == name of the class
- names in this namespace are the class attributes (e.g., class methods)
- The class attributes accessed using the standard namespace notation '.' operator

```
>>> list.pop
<method 'pop' of 'list' objects>
>>> list.sort
<method 'sort' of 'list' objects>
>>> dir(list)
['__add__', '__class__',
...
'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

Function `dir()` : list class attributes



class methods

class method-ը function է սահմանված class-ի namespace-ում.

Երբ Python-ը կատարում է

```
lst.sort()
```

այն սկզբում թարգմանում է

```
list.sort(lst)
```

և կատարում է այս վերջի statement-ը

```
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> lst.sort()
>>> lst
[1, 2, 3, 7, 8, 9]
>>>
```

class methods

class method-ը function է սահմանված class-ի namespace-ում.

Երբ Python-ը կատարում է

```
lst.sort()
```

այն սկզբում թարգմանում է

```
list.sort(lst)
```

և կատարում է այս վերջի statement-ը

```
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> lst.sort()
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> list.sort(lst)
>>> lst
[1, 2, 3, 7, 8, 9]
>>>
```

class methods

class method-ը function է սահմանված class-ի namespace-ում.

Երբ Python-ը կատարում է

```
lst.append(6)
```

այն սկզբում թարգմանում է

```
list.append(lst, 6)
```

և կատարում է այս վերջի statement-ը

```
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> lst.sort()
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> list.sort(lst)
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst.append(6)
>>> lst
[1, 2, 3, 7, 8, 9, 6]
>>>
```

class methods

class method-ը function է սահմանված class-ի namespace-ում.

Երբ Python-ը կատարում է

```
lst.append(6)
```

այն սկզբում թարգմանում է

```
list.append(lst, 6)
```

և կատարում է այս վերջի statement-ը

```
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> lst.sort()
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> list.sort(lst)
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst.append(6)
>>> lst
[1, 2, 3, 7, 8, 9, 6]
>>> list.append(lst, 5)
>>> lst
[1, 2, 3, 7, 8, 9, 6, 5]
```

class methods

class method-ը function է սահմանված class-ի namespace-ում.

Երբ Python-ը կատարում է

```
instance.method(arg1, arg2, ...)
```

այն սկզբում թարգմանում է

```
class.method(instance, arg1, arg2, ...)
```

և կատարում է այս վերջի statement-ը

```
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> lst.sort()
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> list.sort(lst)
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst.append(6)
>>> lst
[1, 2, 3, 7, 8, 9, 6]
>>> list.append(lst, 5)
>>> lst
[1, 2, 3, 7, 8, 9, 6, 5]
```

class methods

class method-ը function է սահմանված class-ի namespace-ում.

Երբ Python-ը կատարում է

```
instance.method(arg1, arg2, ...)
```

այն սկզբում թարգմանում է

```
class.method(instance, arg1, arg2, ...)
```

և կատարում է այս վերջի statement-ը

function-ը ունի extra argument,
որը method-ը հայցող object-ն է

```
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> lst.sort()
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst = [9, 1, 8, 2, 7, 3]
>>> lst
[9, 1, 8, 2, 7, 3]
>>> list.sort(lst)
>>> lst
[1, 2, 3, 7, 8, 9]
>>> lst.append(6)
>>> lst
[1, 2, 3, 7, 8, 9, 6]
>>> list.append(lst, 5)
>>> lst
[1, 2, 3, 7, 8, 9, 6, 5]
```

Exercise 5

Գրեք Python statement օգտագործելով այս գրառումը

`class.method(instance, arg1, arg2, ...)`

և նշեք

`instance.method(arg1, arg2, ...)`

```
>>> s = "To Be Or Not To Be"
>>> s.lower()
'to be or not to be'
>>> s.find("Be")
3
>>> s.replace("Be", "BE")
'To BE Or Not To BE'
>>> s.split()
['To', 'Be', 'Or', 'Not', 'To', 'Be']
>>> ' '.join(s.split())
'To Be Or Not To Be'
>>> '--'.join(s.split())
'To--Be--Or--Not--To--Be'
```


Exercise 5

Գրեք Python statement օգտագործելով այս գրառումը

`class.method(instance, arg1, arg2, ...)`

և նշեք

`instance.method(arg1, arg2, ...)`

```
>>> s = "To Be Or Not To Be"
>>> s.lower()
'to be or not to be'
>>> s.find("Be")
3
>>> s.replace("Be", "BE")
'To BE Or Not To BE'
>>> s.split()
['To', 'Be', 'Or', 'Not', 'To', 'Be']
>>> ' '.join(s.split())
'To Be Or Not To Be'
>>> '--'.join(s.split())
'To--Be--Or--Not--To--Be'
```

```
>>> str.lower(s)
'to be or not to be'
>>> str.find(s, 'Be')
3
>>> str.replace(s, 'Be', 'BE')
'To BE Or Not To BE'
>>> str.split(s)
['To', 'Be', 'Or', 'Not',
 'To', 'Be']
>>> str.join('--', s.split())
'To--Be--Or--Not--To--Be'
```

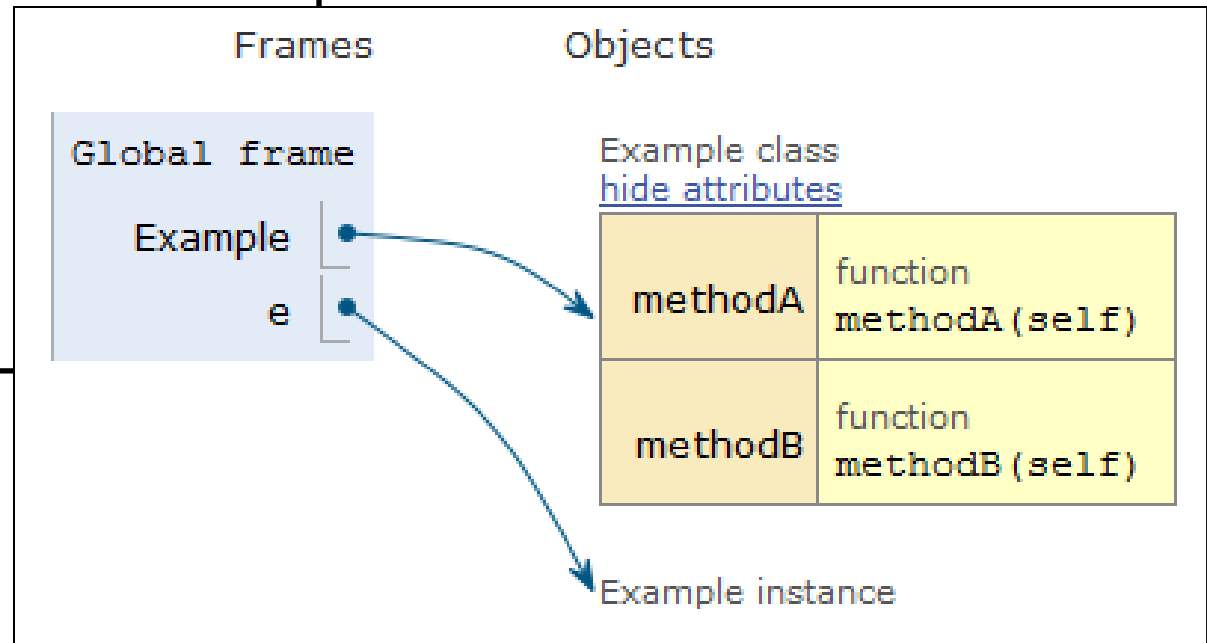
References

1. Franek. “CS 1MD3 Introduction to Programming.” Accessed July 8, 2014.
2. “The Python Tutorial — Python 3.4.1 Documentation.” Accessed August 2, 2014.
<https://docs.python.org/3.4/tutorial/>.
3. “Recursion — Problem Solving with Algorithms and Data Structures.” Accessed December 19, 2014.
<http://interactivepython.org/courselib/static/pythonds/Recursion/recursionsimple.html>.

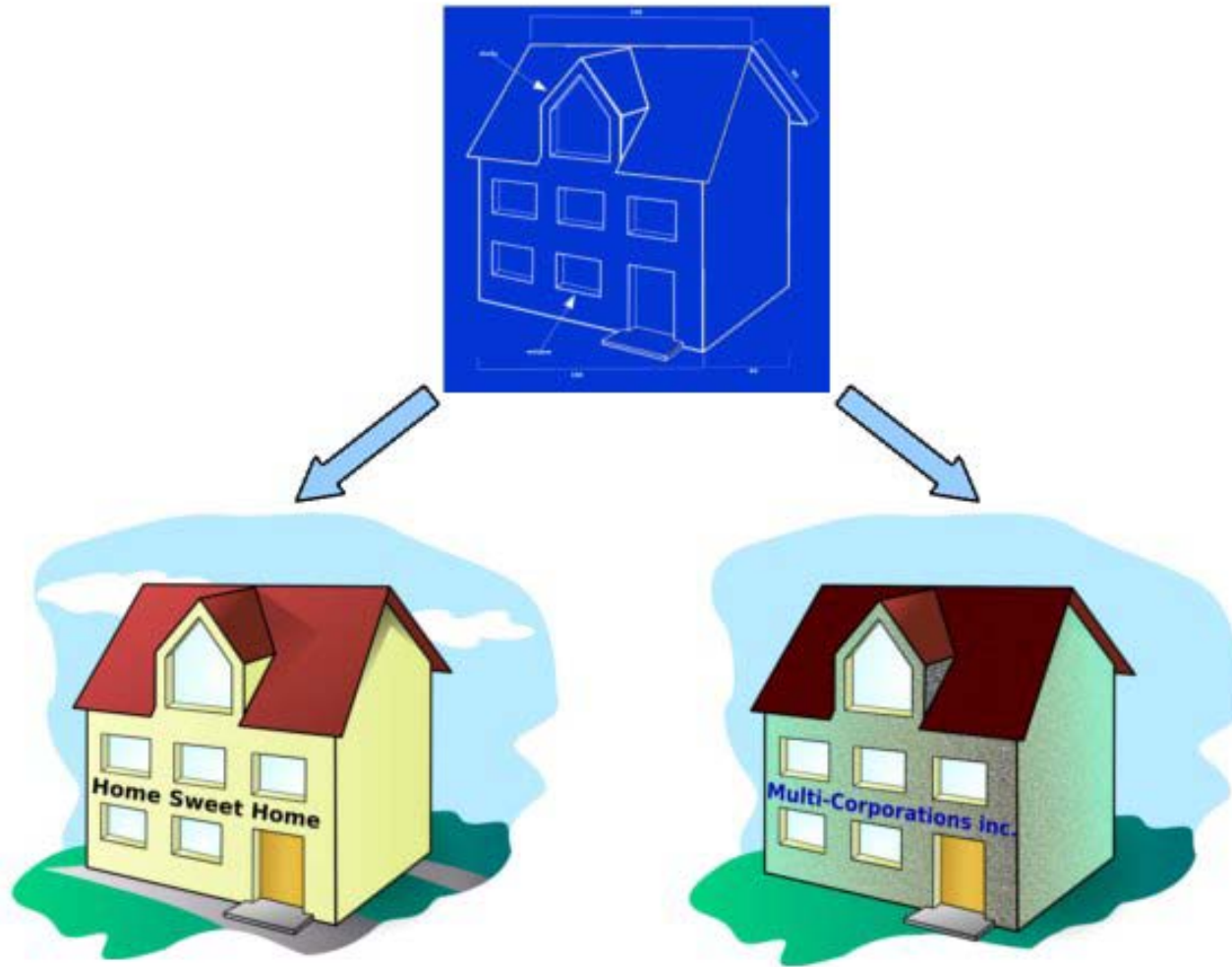
empty class

```
class Example():  
    def methodA(self):  
        pass  
  
    def methodB(self):  
        pass
```

```
e = Example()  
e.methodA()  
e.methodB()
```



Classes and Objects



Classes and Objects

