

ОСНОВИ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ, НЕЙРОННИХ МЕРЕЖ ТА ГЛИБОКОГО НАВЧАННЯ

Частина 6. ВИСОКОРІВНЕВА МОВА ПРОГРАМУВАННЯ PYTHON

Лекція 6.4. Функціональне програмування.

ФУНКЦІЇ

Функція → базова програмна структура мови , що забезпечує багаторазове використання програмного коду і зменшує його надмірність.


Функція → засіб, що дозволяє групувати набори інструкцій так, що в програмі вони можуть запускатися неодноразово.

ФУНКЦІЇ

Інструкція	Приклад
def , return	<pre>def myfunc(a,b): return a+b</pre>
Виклик	<pre>myfunc(1,2)</pre>
yield	<pre>def sqr(x): for i in range (x): yield i**2</pre>
global	<pre>def spb(): global x; x = 'new'</pre>
nonlocal	<pre>def spb(): nonlocal x; x = 'new'</pre>
lambda	<pre>func = [lambda x: x*2, lambda x: x**2]</pre>

СТВОРЕННЯ ФУНКЦІЇ

def <name>(arg1, arg2,...,argN) : Заголовок



<statements>

Відступ




def <name>(arg1, arg2,...,argN) :

<statements>

return <value>

Вертає результат
(необов'язково)



<statements>

def - **ІНСТРУКЦІЯ**, яка створює новий об'єкт типу **функція** і присвоює ім'я цьому об'єкту.

ПОЛІМОРИФІЗМ

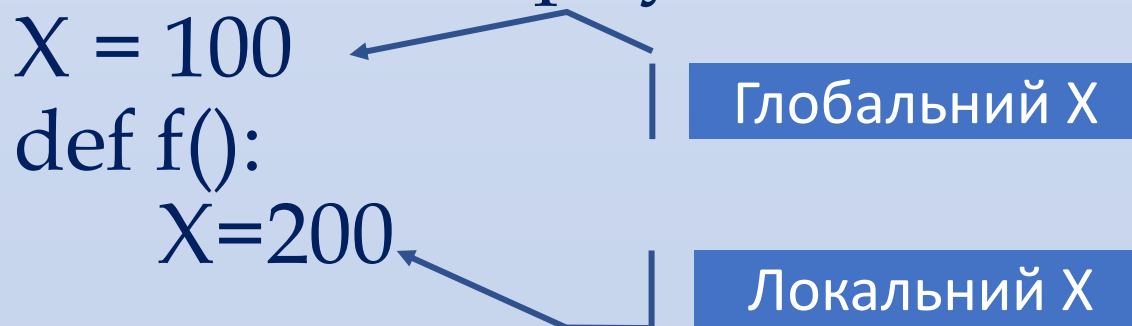
!!! ПОЛІМОРИФІЗМ – сенс операції залежить від типів оброблюваних об'єктів. Python - це мова з динамічною типізацією, поліморфізм в ньому проявляється всюди. Всі операції в мові Python є поліморфічні, поки об'єкти підтримують очікуваний інтерфейс (протокол), функція зможе обробляти їх.

Поліморфізм - це можливість обробки різних типів даних за допомогою "однієї і тієї ж" функції, або методу.

ОБЛАСТЬ ВИДИМОСТІ

За замовчуванням всі імена, значення яких присвоюються всередині функції, асоціюються з простором імен цієї функції і ніяк інакше. Це **ЛОКАЛЬНІ** змінні.

- Імена, які визначаються всередині інструкції **def**, видно тільки програмному коду всередині інструкції **def**. До цих імен **неможна** звернутися за межами функції.



ОБЛАСТЬ ВИДИМОСТІ

- **LOCAL** область - присвоювання змінної виконується всередині інструкції *def*, змінна є локальною для цієї функції.
 - **NONLOCAL** область - присвоювання проводиться в межах охоплюючої інструкції *def*, змінна є нелокальною для цієї функції.
 - **GLOBAL** область присвоювання проводиться за межами всіх інструкцій *def*, вона є глобальною для всього файлу.
- Python - **лексична область видимості** - видимість змінної визначаються місцем розташування цієї змінної у вихідних текстах програми, а не місцем, звідки викликаються функції.

ІНСТРУКЦІЇ **global**, **nonlocal**

!!! Інструкції **global** / **nonlocal** не оголошують тип або розмір змінної - вони оголошують простір імен.

Інструкція **global** дозволяє змінювати змінні, що знаходяться на верхньому рівні модуля, за межами інструкції **def**.

- Глобальні імена - це імена, які визначені на верхньому рівні модуля, що вміщує їх.
- Глобальні імена повинні оголошуватися, тільки якщо їм будуть присвоюватися значення всередині функцій.
- Звертатися до глобальних імен всередині функцій можна і без об'явлення їх глобальними.

Інструкція **nonlocal** застосовується до імен, які перебувають в локальних областях видимості охоплюючої інструкції **def**.

ПРАВИЛО LEGB

PYTHON Вбудована область видимості

Зумевлені імена (open, range)

B

МОДУЛЬ Глобальна область видимості

Імена, визначені на верхнім рівні модуля або оголошені в інструкції **def** як глобальні

G

ФУНКЦІЯ non локальна область видимості

Імена, визначені в локальній області видимості та в функції, яка охоплює

E

ФУНКЦІЯ локальна області видимості

Імена, визначені в тілі функції

L

ОБЛАСТЬ ВИДИМОСТИ

```
X = 100
def func(Y):
    Z=X+Y
    return Z
print (func(1))
print (X)
```

101
100

```
X = 100
def func(Y):
    X=200
    Z=X+Y
    return Z
print (func(1))
print (X)
```

201
100

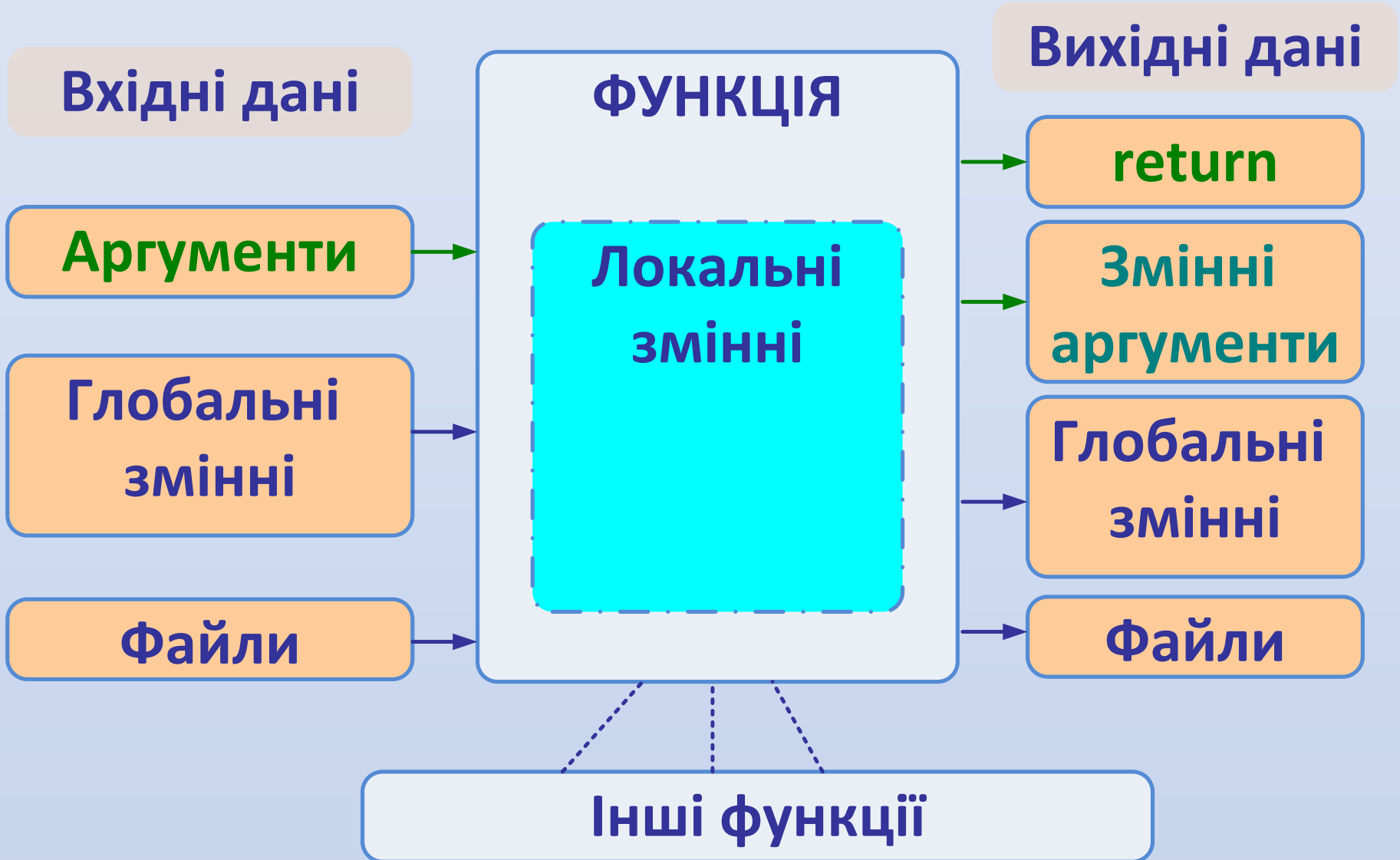
```
X = 100
def func(Y):
    global X
    X=200
    Z=X+Y
    return Z
print (func(1))
print (X)
```

201
200

АРГУМЕНТИ

- **НЕЗМІННІ АРГУМЕНТИ** - передаються «за значенням» – передаються в вигляді **посилань** на об'єкти – (не в вигляді копій!).
Безпосередня зміни аргументу всередині функції НЕМОЖЛИВА.
- **ЗМІННІ АРГУМЕНТИ** - передаються «за вказівником» – передаються в вигляді **вказівника** . Допускається можливість **безпосередньої зміни аргументу всередині функції.**

АРГУМЕНТИ



АРГУМЕНТ за ЗАМОВЧУВАННЯМ

Дозволяють зробити окремі аргументи функції необов'язковими.

Якщо значення не передається, аргумент отримує значення за замовчуванням, яке визначено у заголовку функції

```
def <name>(arg1, arg2=<def_value>,...) :  
    <statements>
```

АРГУМЕНТИ. РЕКОМЕНДАЦІЇ

- Для передачі значень до функції використовуйте аргументи, для повернення результатів - інструкцію **return**.
- **Не використовуйте глобальні змінні** (! тільки якщо це дійсно необхідно).
- **Не впливайте на змінювані аргументи**, якщо модуль, що викликає, не передбачає цього.

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

#Визначення функції

```
def foo(arg1, arg2,...,argN) :
```

<statements>

#Виклик функції

```
foo(par_x, par_y,..., par_z)
```



Який параметр виклику відповідає аргументу визначення ?

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

- **ПОЗИЦІЙНІ** аргументи - відповідність визначається за позицією аргументу у визначенні функції та її виклику.
- **ІМЕНОВАНІ** аргументи – дозволяють визначити відповідність за іменами, а не за позиціями аргументів.

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

Опис	Виклик	
<code>def func(name)</code>	<code>func(value)</code>	За позицією або за ім'ям
<code>def func (name=value)</code>	<code>func (name=value)</code>	Значення аргументу за замовчуванням, якщо аргумент не передається функції
<code>def func(*name)</code>	<code>func (*sequence)</code>	Довільне число аргументів за позицією (кортеж)
<code>def func(**name)</code>	<code>func(** dict)</code>	Довільне число аргументів за іменами (словник)
<code>def func(*args, name)</code>		≥3.0 Аргументи, що передаються тільки за іменами
<code>def func(*, name=value)</code>		

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

У заголовку функції аргументи повинні вказуватися в наступному порядку:
будь-які звичайні аргументи (**name**),
- за якими можуть слідувати аргументи зі значеннями за замовчуванням (**name = value**),
-- за якими слідують аргументи в формі ***args** (або ***** в 3.0), якщо є ,
--- за якими можуть слідувати будь-які імена або пари **name = value** аргументів, які передаються тільки по імені (в 3.0),
---- за якими можуть слідувати аргументи в формі ****kwargs** (останні!).

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

У виклику функції аргументи повинні вказуватися в наступному порядку: будь-які позиційні аргументи (значення), за якими можуть слідувати будь-які іменовані аргументи (**name = value**) і аргументи у формі ***sequence**, за якими можуть слідувати аргументи в формі ****dict** (остання, відповідає ****kwargs** заголовку).

ЗІСТАВЛЕННЯ АРГУМЕНТІВ

Дії інтерпретатору:

1. Сопоставлення неіменованих аргументів за позиціями.
2. Сопоставлення іменованих аргументів за іменами
3. Сопоставлення додаткових неіменованих аргументів з кортежем **args*.
4. Сопоставлення додаткових іменованих аргументів з словником ***kwargs*.
5. Сопоставлення значень за замовчуванням з відсутніми іменованими

РЕКУРСИВНА ФУНКЦІЯ

Визначено деяке x_0 .

Обчислення x_n -будується як $x_i = F(x_{i-1})$,
 $i=1,2,\dots, n$.

Кожне обчислення $F(x)$ – ітерація, i –
номер ітерації, процес – *ітеративний*.

З іншої сторони : $x_n = F(F(F(\dots F(x_0)))$ –
рекурентний процес обчислення x_n .

Рекурсія → – метод визначення об'єкту
через раніш визначені об'єкти, серед яких
сам об'єкт.

Ітерація – багаторазове повторення.

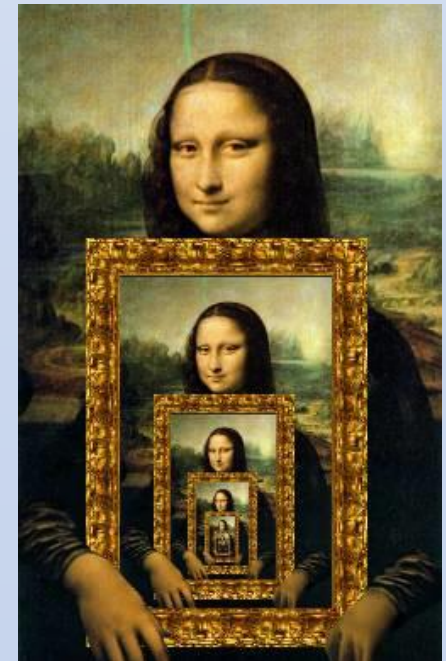
Рекурсія – багаторазове звернення.

РЕКУРСИВНА ФУНКЦІЯ

Теорема → рекурсивне обчислення завжди можна перетворити в ітераційне обчислення (що використовує цикли). І навпаки, будь-яке ітераційне обчислення, що припускає використання циклів, можна реалізувати як рекурсивне.

Рекурсія VS Ітерація → предмет багаторічних суперечок програмістів

	Рекурсія	Ітерація
Запис	Компактний	НЕ компактний
Час	Повільніше	Швидше
Пам'ять	Більше	Менше



РЕКУРСИВНА ФУНКЦІЯ

Рекурсія → – метод визначення об'єкту через раніш визначені об'єкти, серед яких сам об'єкт.

Рекурсивна функція → – це така функція, серед виконуваних інструкція, якою є оператор виклику самої цієї функції.

```
def <name>(arg1, arg2,...,argN) :  
    <statements>  
    name (_arg1_, _arg2_, ... _argN_)  
    <statements>
```

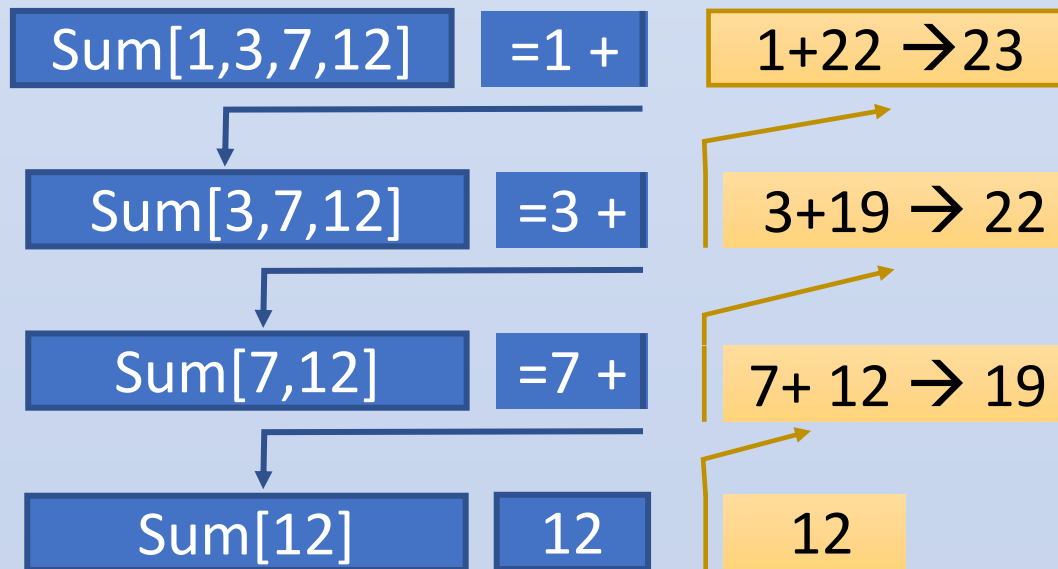
РЕКУРСИВНА ФУНКЦІЯ

```
def listsum (L) :  
    if len(L)==1:  
        return L[0]  
    else:  
        return L[0]+listsum(L[1 :])
```

Нерекурсивна гілка
(вихід з рекурсії, база)

Рекурсивна гілка (тіло)

Print(listsum([1,3,7,12]))



ІНТРОСПЕКЦІЯ

ІНТРОСПЕКЦІЯ – можливість для будь-якого об'єкта (функції також) отримати всю інформацію про його внутрішню структуру і середовищі виконання.

Дві групи: а) стандартні можливості (описані в документації по мові),
б) нестандартні (характерні для конкретної реалізації мови (наприклад, *CPython*)).

АНОТАЦІЇ

Анотації (короткий опис) не мають ніякого семантичного значення, використовуються в Python тільки для підтримки інформативності коду та його автоматизованого аналізу. Анотування це опція , не вимога мови.

```
def <name>(arg1 : expr, \
            arg2 : expr = value, \
            ..., *args : expr \
            *kwargs : expr ) : ->expr
    <statements>
```



АНОТАЦІЇ

Доступ до анотації через атрибут функції
`__annotations__`

Вертає словник.

```
def foo(a: 'x', b: 5 + 6, c: list) -> max(2, 9): ...
```

```
foo.__annotation__
```

```
{'a': 'x', 'b': 11, 'c': list, 'return': 9}
```

АНОНІМНІ ФУНКЦІЇ (*lambda*)

Створення об'єкту «функція» в формі виразу.

Lambda – це **вираз**!

Lambda - складається з **одного** виразу!

Lambda - вираз вертає функцію, але **НЕ зв'язує** створений об'єкт (функцію) з іменем (змінною).

Головне: можна використовувати там, де def неможливо. Наприклад, в інших виразах.

LAMBDA ВИРАЗ

lambda_expr ::=

lambda **arg1, arg2,...,argN : some_expression**

Lambda вираз

def lamda (arg1, arg2,...,argN) :
***return* some_expression**

Еквівалентна функція

Lambda - вираз не може містити анотацій та інші вирази.

Аргументи та області видимості аналогічні def.../

ВІДОБРАЖЕННЯ НА ПОСЛІДОВНІСТЬ (map)

map_function ::=

map (**func**, iterable1, iterable2, ... iterableN)

def func(arg1, arg2, ... argN)



Застосовує **func** до кожного елементу ітеруємої послідовності/ послідовностей.

python ver < 3.0 -> list

python ver 3.0+ -> iterator

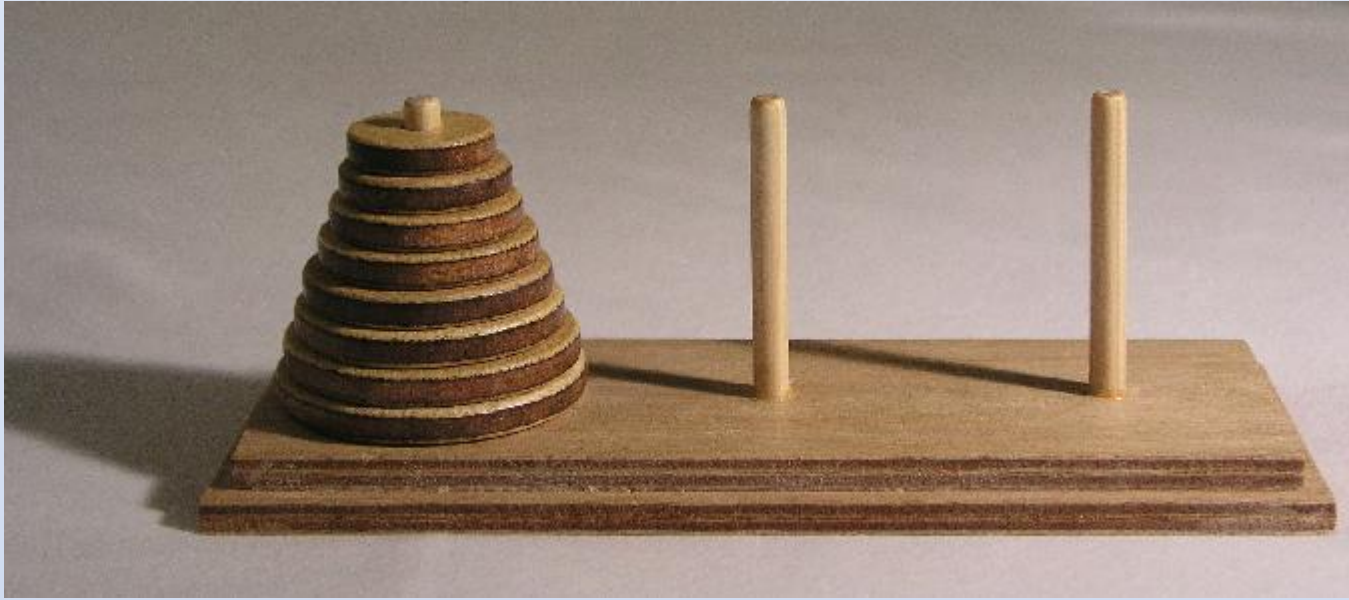
ВІДОБРАЖЕННЯ НА ПОСЛІДОВНІСТЬ (map)

Не рекомендовано

! filter () ver < 3.0

! reduce () ver < 3.0

ХАНОЙСЬКІ ВЕЖІ



Дано три стрижня, на один з яких нанизані вісім кілець (n), причому кільця відрізняються розміром і лежать менший на більший. Завдання полягає в тому, щоб перенести піраміду з восьми кілець за найменше число ходів на інший стрижень. За один раз дозволяється переносити тільки одне кільце, причому не можна класти більше кілець на меншу.

ХАНОЙСЬКІ ВЕЖІ



Написати програму з використанням рекурсивного виклику функції.

Рекомендована ЛІТЕРАТУРА

- **Програмування числових методів мовою Python:** підруч. / А. В. Анісімов, А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2014. – 640 с.
- **Програмування числових методів мовою Python:** навч. посіб. / А. Ю. Дорошенко, С. Д. Погорілий, Я. Ю. Дорогий, Є. В. Глушко ; за ред. А. В. Анісімова. – К. : Видавничо-поліграфічний центр "Київський університет", 2013. – 463 с.
- **Основи програмування Python:** Підручник для студ. спеціальності 122 «Компютерні науки» / А.В.Яковенко; КПІ.- Київ: КПІ, 2018 . – 195 с.
- **Лутц М.** Изучаем Python, 4-е издание. - СПб.: Символ-Плюс. 2011.- 1280 с.: ил.

Контрольні запитання

- Наведіть визначення функції в мові Python. Надайте опис інструкції створення функції. вкажіть призначення її компонентів.
- Наведіть перелік областей видимості для змінних функції. Надайте інструкції керування областями видимості для змінних функції. Наведіть приклади використання.
- Наведіть визначення аргументів за замовчуванням, наведіть приклади використання.
- Наведіть правила зіставлення аргументів функції у її описі та її виклику.
- Наведіть визначення рекурсивної функції в мові Python. Надайте приклади створення та використання рекурсивних функцій.
- Визначте поняття анотації функції, надайте приклади створення анотацій та їх використання.
- Наведіть визначення `lambda` виразу, надайте приклади створення та використання **lambda** виразів.

The END

Частина 6. Лекція 6.4.