

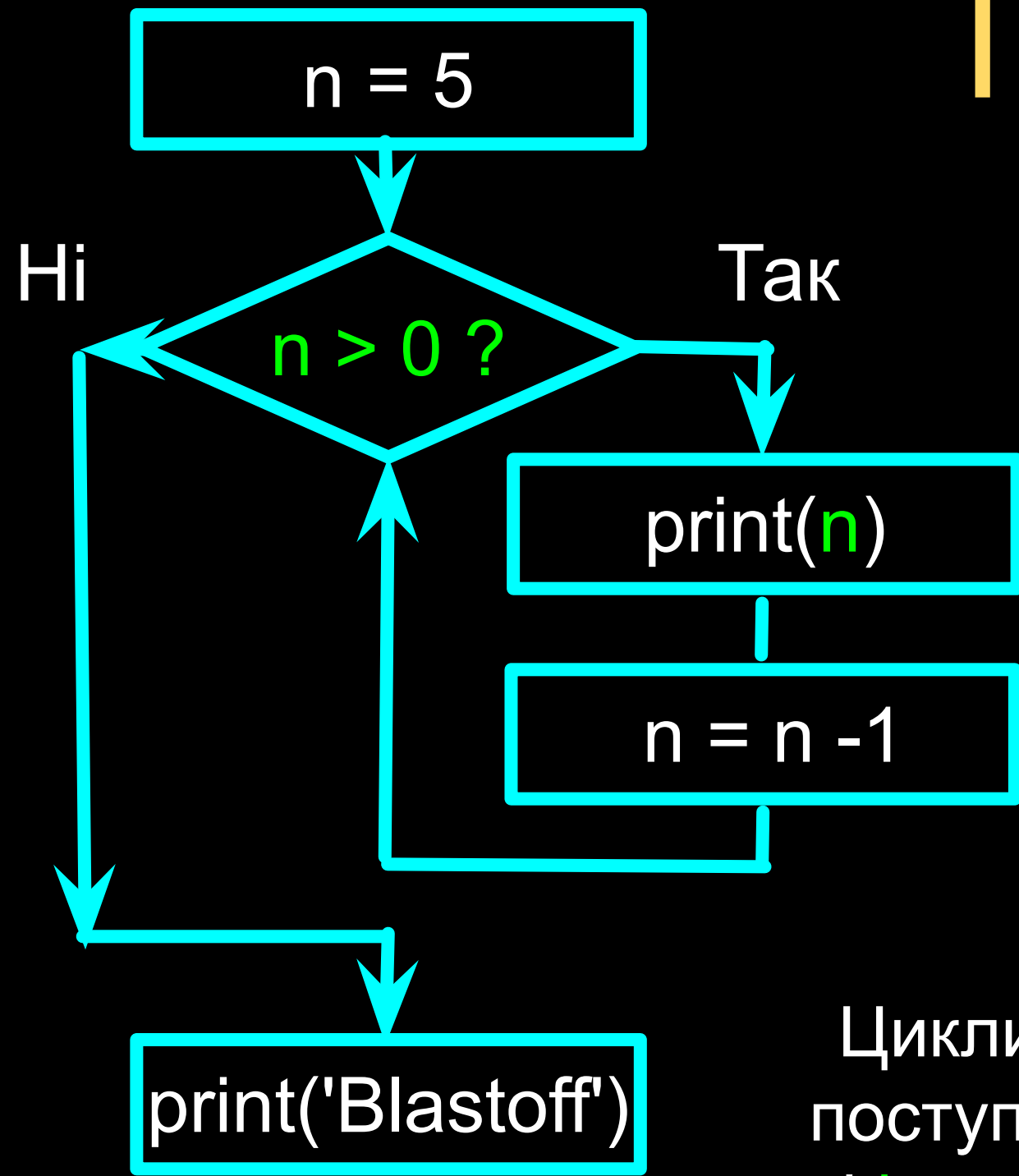
Цикли та ітерації

Розділ 5

Python для всіх
www.py4e.com



Повторювані кроки



Програма:

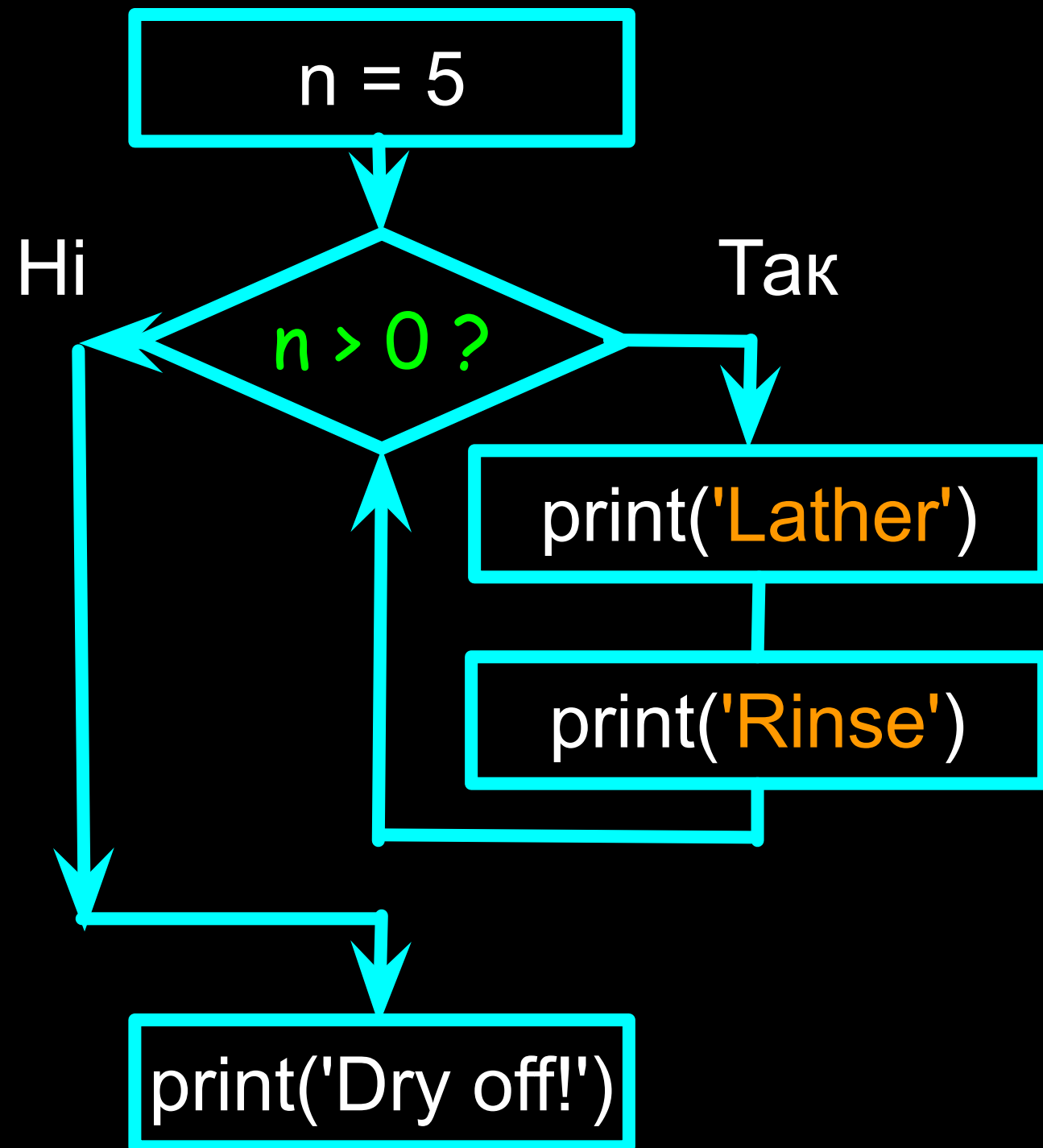
```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Blastoff!')
print(n)
```

Вивід:

5
4
3
2
1
Blastoff!
0

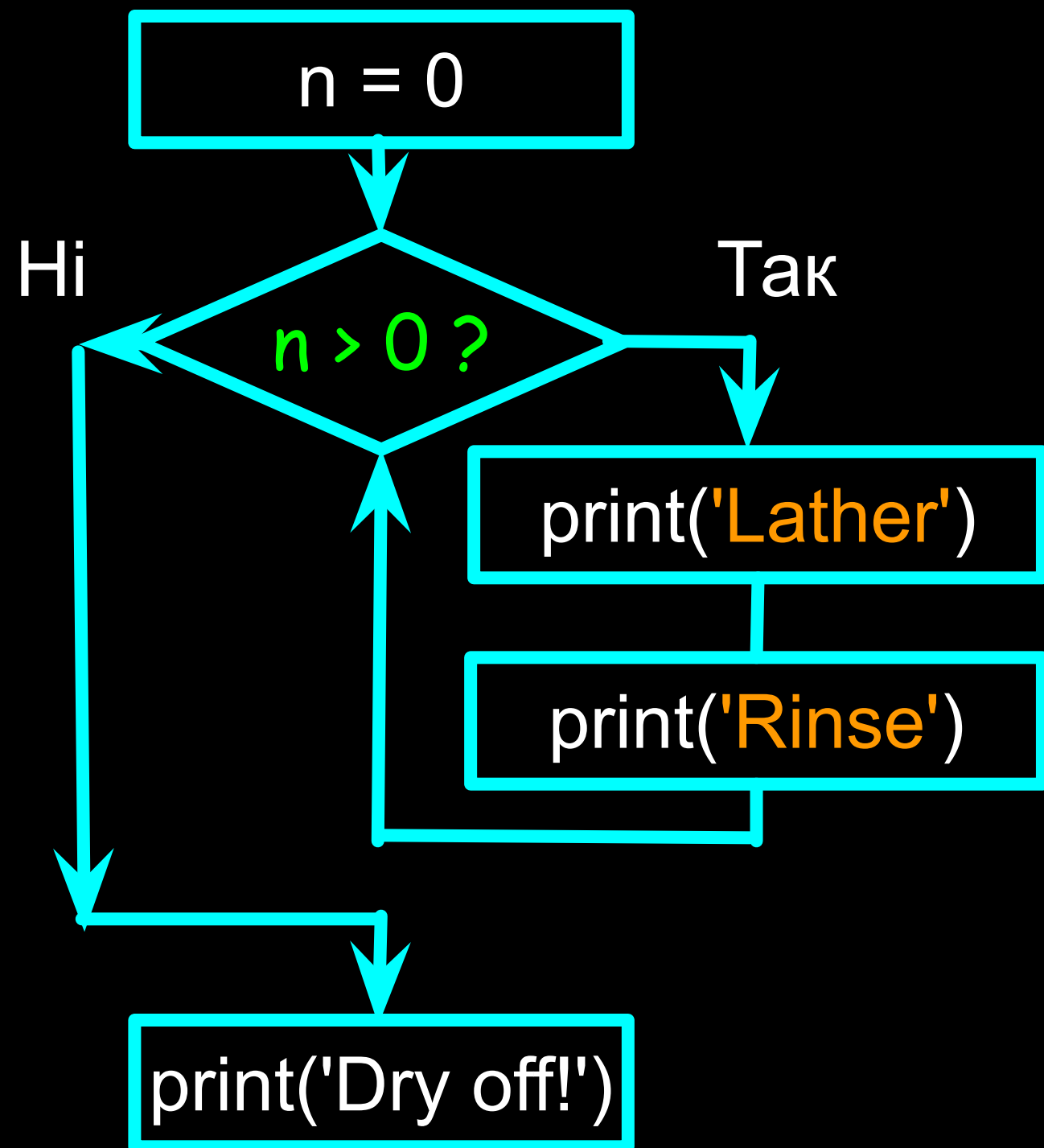
Цикли (повторювані кроки) мають **ітераційні змінні**, які поступово змінюються кожну **ітерацію** в циклі. Зазвичай ці **ітераційні змінні** задаються через послідовність чисел

Нескінченний цикл



```
n = 5
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

Що не так з цим циклом?



ІНШИЙ ЦИКЛ

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
print('Dry off!')
```

Що робить цей цикл?

Вихід з циклу

- Інструкція **break** завершує поточний цикл, програма перестрибує до рядка, що іде одразу після циклу
- Це схоже на тестування циклу, яке можна здійснити будь-де в його тілі


```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> finished
finished
> done
Done!
```

Вихід з циклу

- Інструкція **break** завершує поточний цикл, програма перестрибує до рядка, що іде одразу після циклу
- Це схоже на тестування циклу, яке можна здійснити будь-де в його тілі

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```

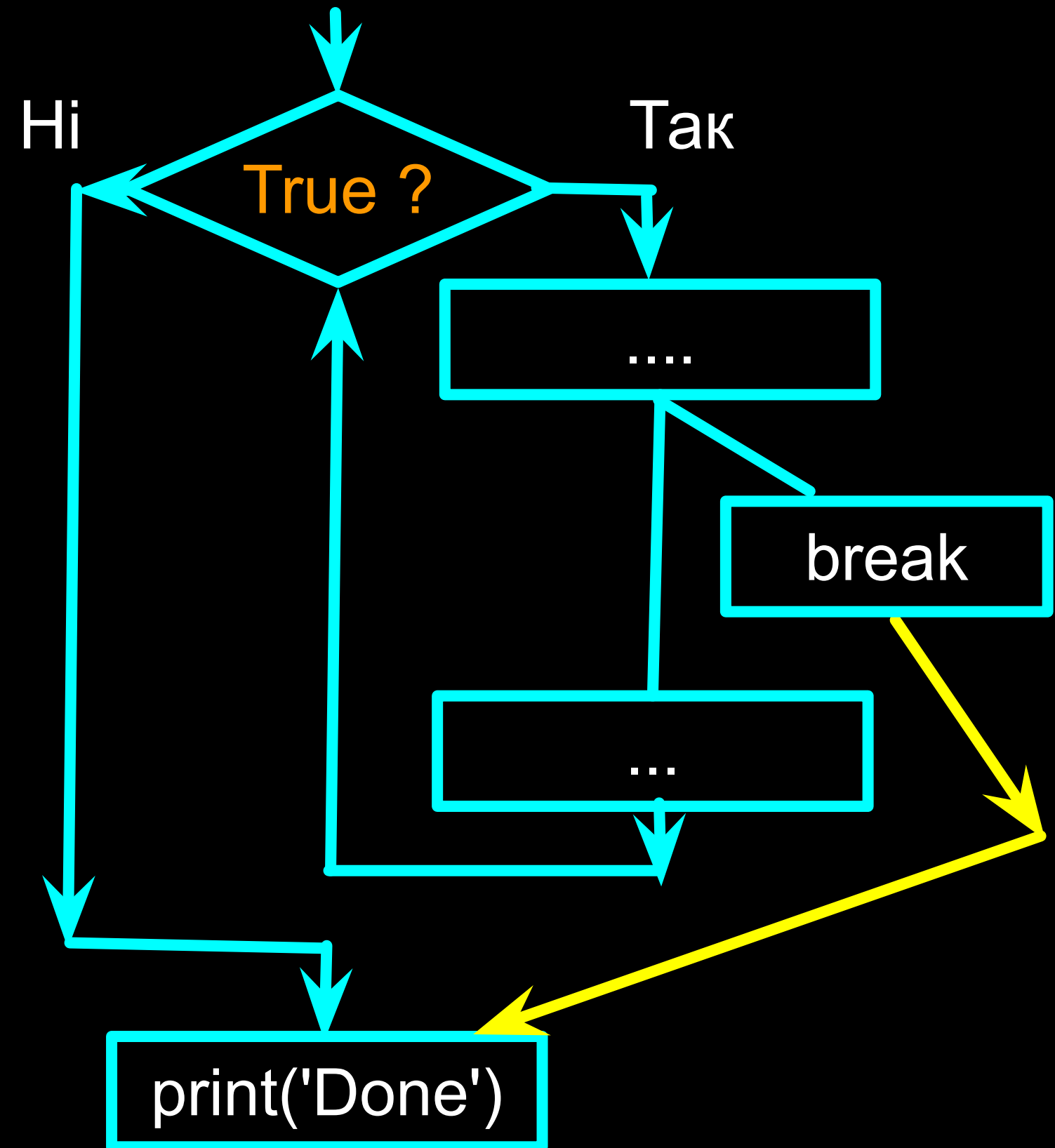


```
> hello there
hello there
> finished
finished
> done
Done!
```

```
while True:
    line = input('> ')
    if line == 'done' :
        break
    print(line)
print('Done!')
```



[http://en.wikipedia.org/wiki/Transporter_\(Star_Trek\)](http://en.wikipedia.org/wiki/Transporter_(Star_Trek))



Завершення ітерації з продовженням (continue)

Інструкція **continue** завершує поточну ітерацію, тож програма перестрибує вгору циклу та розпочинає наступну ітерацію

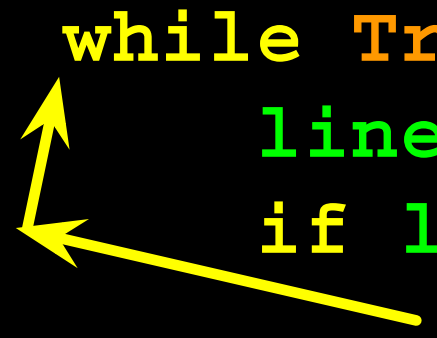
```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```


Завершення ітерації з продовженням (continue)

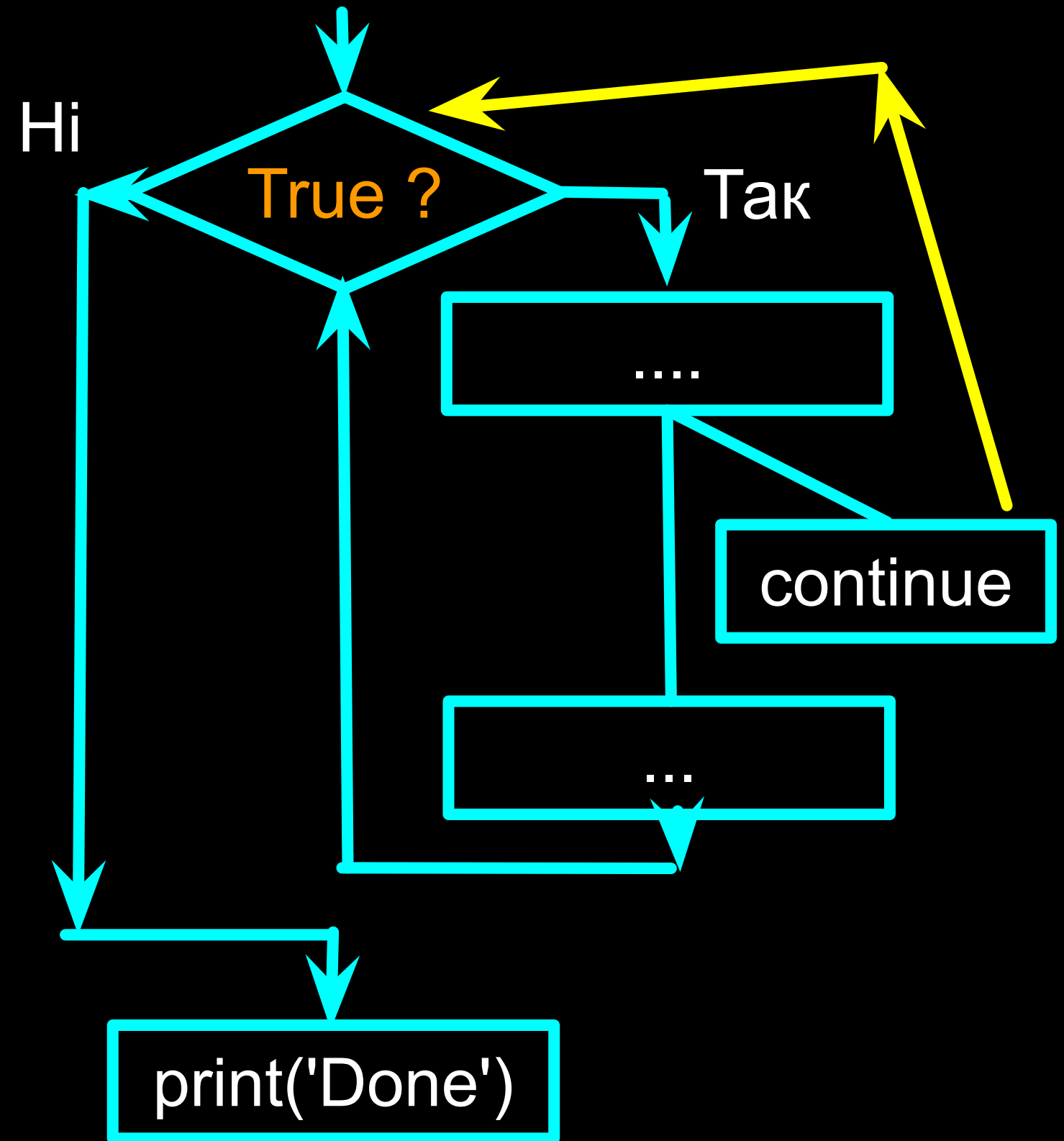
Інструкція `continue` завершує **поточну ітерацію**, тож програма перестрибує **вгору циклу** та розпочинає наступну ітерацію

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```



```
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!
```

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```



Невизначені цикли

- Цикли while називаються **«невизначеними циклами»**, бо вони продовжують працювати допоки логічна умова не стане **хибною (False)**
- Цикли, які ми бачили донині, досить легко дослідити, щоби зрозуміти, чи завершаться вони, чи стануть «нескінченними»
- Часом буває трохи складніше визначити, чи завершиться цикл

Визначені цикли

Перебір набору елементів...

Визначені цикли

- Досить часто ми маємо **список** елементів у **рядках файлу** – фактично скінченну **множину речей**
- За допомогою інструкції **for** ми можемо створити в Python цикл, який виконуватиметься один раз для кожного з цих елементів
- Ці цикли називаються **«визначеними циклами»**, бо мають чітку скінченну кількість повторів
- Вважається, що **«визначені цикли проходять (перебирають) усі елементи множини»**

Простий визначений цикл

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

5

4

3

2

1

Blastoff!

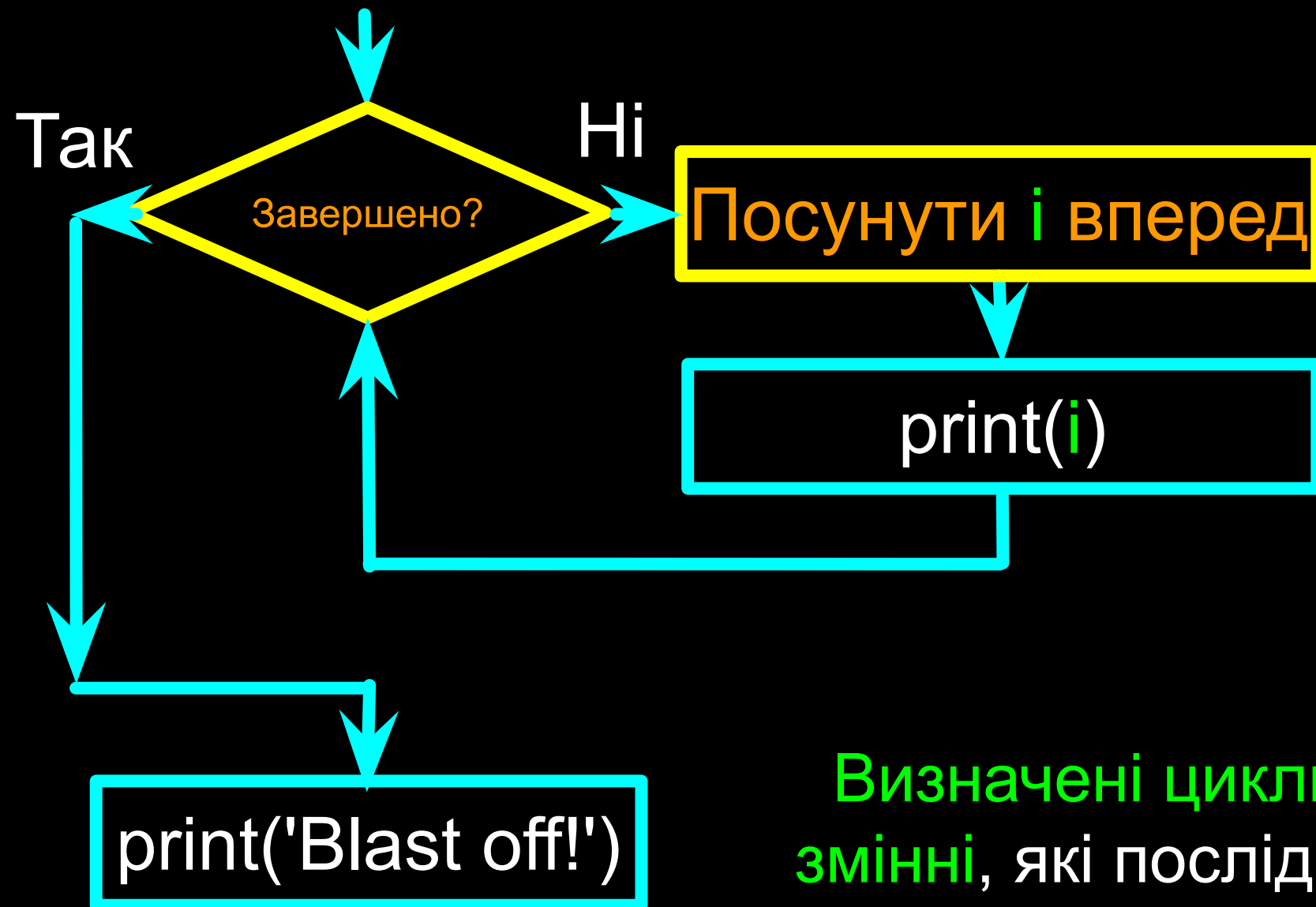
Визначений цикл з рядками

```
friends = ['Joseph', 'Glenn', 'Sally']  
for friend in friends :  
    print('Happy New Year:', friend)  
print('Done!')
```

Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally

Done!

Простий визначений цикл



```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

5
4
3
2
1
Blastoff!

Визначені цикли (цикли for) мають явні ітераційні змінні, які послідовно змінюються на кожній ітерації циклу. Ці ітераційні змінні перебирають визначену послідовність або множину елементів.

Розгляньмо «in»

- Ітераційна змінна
«перебирає» послідовність
(впорядковану множину)
- Блок (тіло) коду
виконується один раз для
кожного елемента в («in»)
послідовності
- Ітераційна змінна
перебирає усі значення в
(«in») послідовності

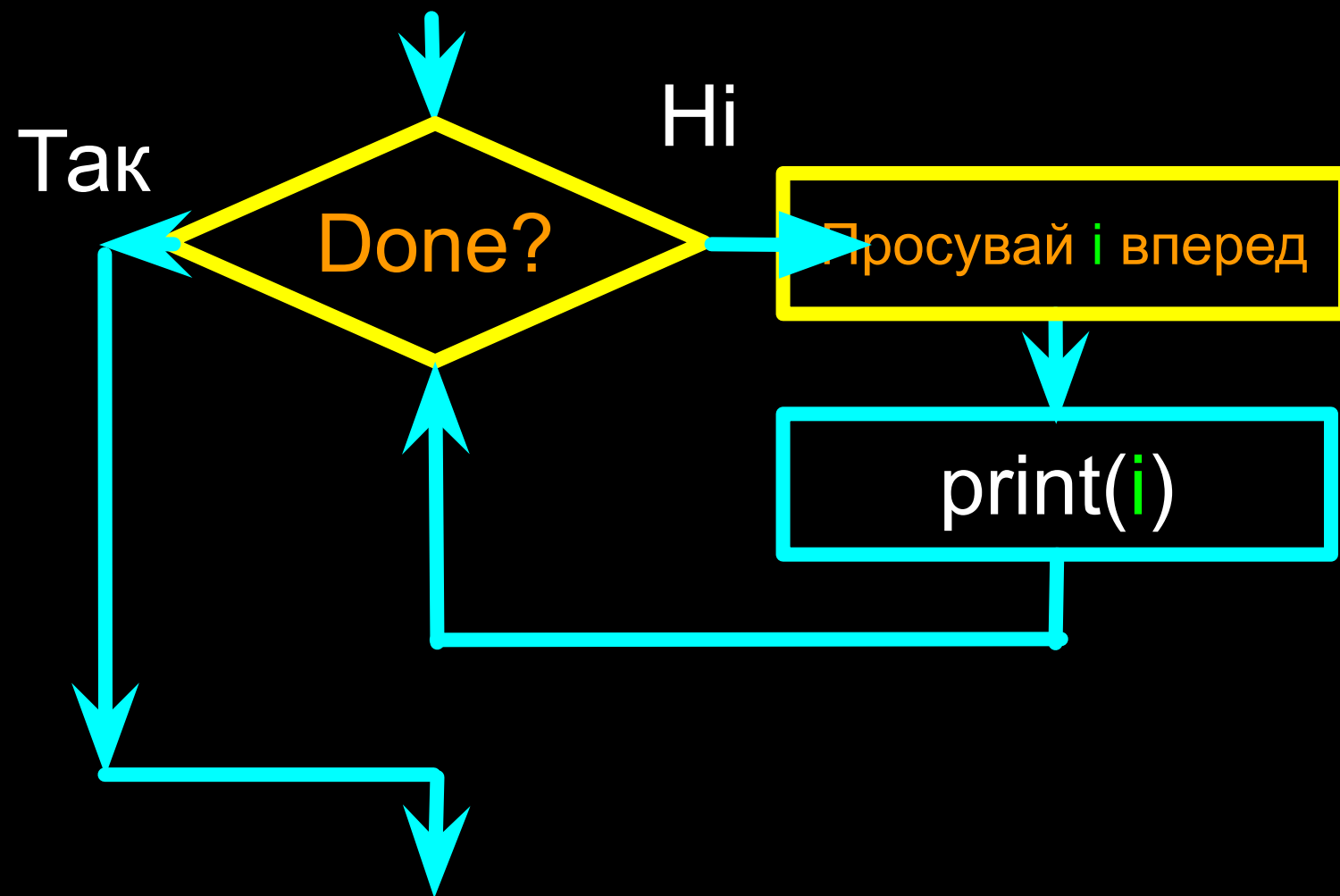
Ітераційна змінна



Послідовність з
п'яти елементів

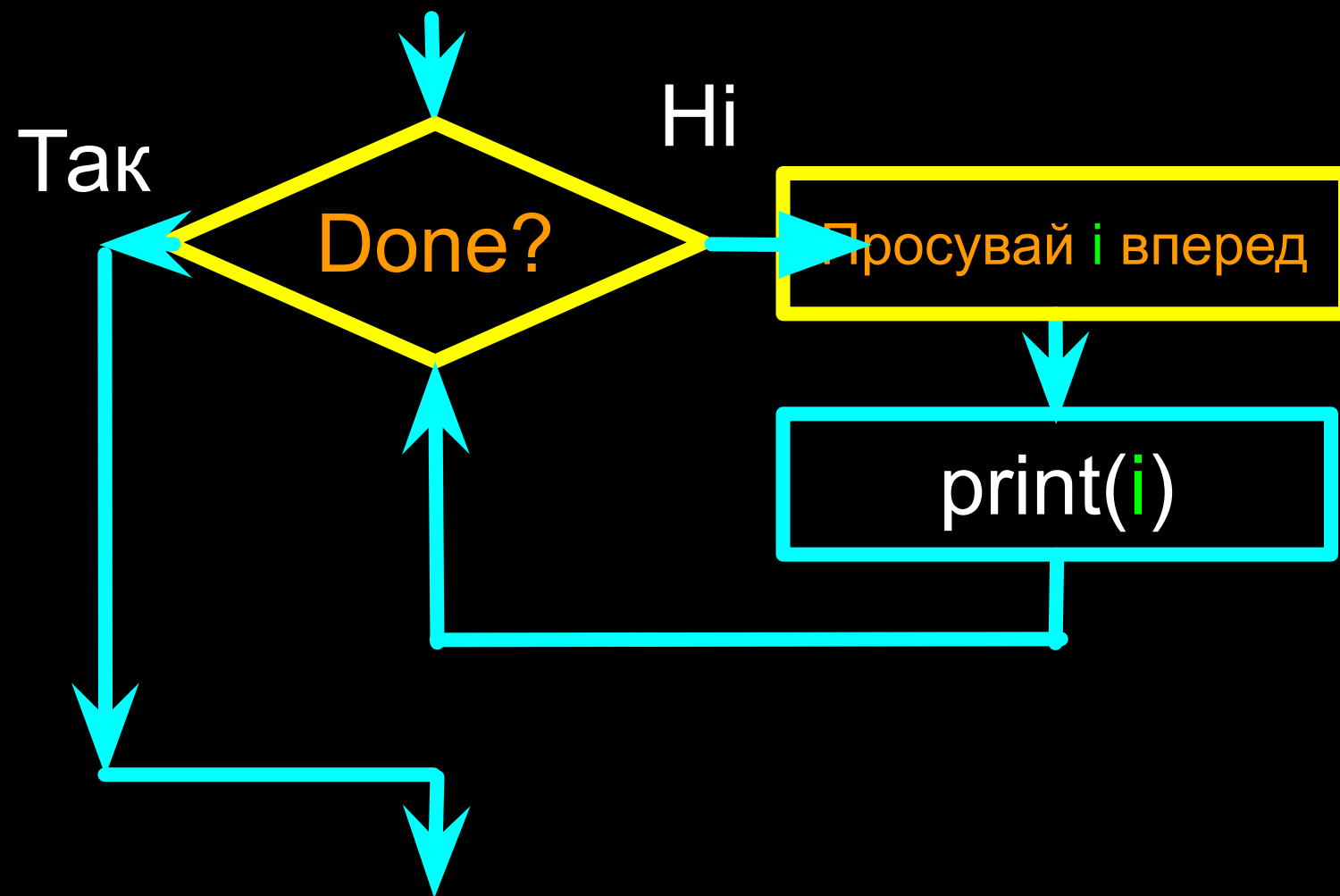


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

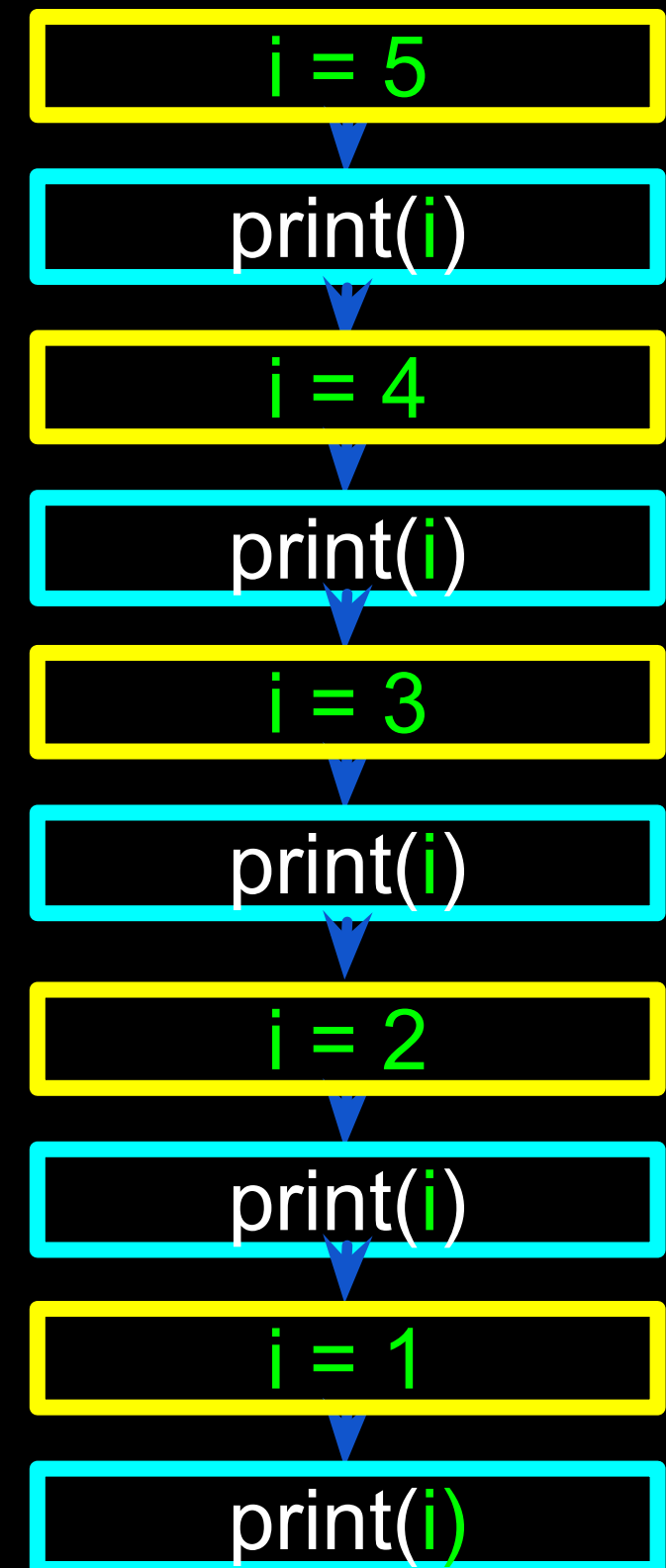


```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```

- Ітераційна змінна «перебирає» послідовність (впорядковану множину)
- Блок (тіло) коду виконується один раз для кожного елементу в («in») послідовності
- Ітераційна змінна перебирає усі значення («in») послідовності



```
for i in [5, 4, 3, 2, 1] :  
    print(i)
```



Ідіоми циклів: Що ми робимо всередині циклів

Примітка: Зауважте: хоча ці приклади досить прості, шаблони можна застосувати до всіх видів циклів

Створюємо «розумні» цикли

Хитрість в тому, щоб «знати» щось про весь цикл, коли ви застрягли в написанні коду, який бачить тільки один запис за раз

Встановіть певним змінним початкові значення

for thing in data:

Шукайте щось або робіть щось для кожного запису окремо, та оновлюйте змінні

Подивіться на змінні

Проходження циклом по МНОЖИНІ

```
print('Before')
for thing in [9, 41, 12, 3, 74, 15] :
    print(thing)
print('After')
```

\$ python basicloop.py

Before

9

41

12

3

74

15

After

Яке число найбільше?

Яке число найбільше?

3

Яке число найбільше?

41

Яке число найбільше?

12

Яке число найбільше?

9

Яке число найбільше?

74

Яке число найбільше?

15

Яке число найбільше?

Яке число найбільше?

3

41

12

9

74

15

Яке число найбільше?

largest_so_far

-1

Яке число найбільше?

3

largest_so_far

3

Яке число найбільше?

41

largest_so_far

41

Яке число найбільше?

12

largest_so_far

41

Яке число найбільше?

9

largest_so_far

41

Яке число найбільше?

74

largest_so_far

74

Яке число найбільше?

15

74

Яке число найбільше?

3 41 12 9 74 15

74

Пошук найбільшого значення

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

\$ python largest.py

Before -1

9 9

41 41

41 12

41 3

74 74

74 15

After 74

Ми створюємо змінну з **найбільшим значенням**, що ми бачили донині. Якщо поточне число, яке ми бачимо, більше, то це нове **найбільше значення, що ми бачили донині**.

Більше шаблонів циклів

Підрахунки у циклі

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
```

```
Before 0
```

```
1 9
```

```
2 41
```

```
3 12
```

```
4 3
```

```
5 74
```

```
6 15
```

```
After 6
```

Щоб **порахувати**, скільки разів ми виконуємо цикл, задаємо **змінну-лічильник**, яка починається з 0, і додаємо до неї **одиницю** щоразу, коли **проходимо цикл**.

Підрахунок суми у циклі

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
```

```
Before 0
```

```
9 9
```

```
50 41
```

```
62 12
```

```
65 3
```

```
139 74
```

```
154 15
```

```
After 154
```

Щоб додати значення в циклі, ми задаємо змінну суми, яка починається з 0, і додаємо до неї значення (збільшуємо її) щоразу, коли проходимо ЦИКЛ.

Знаходження середнього значення в циклі

```
count = 0
sum = 0
print('Before', count, sum)
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print(count, sum, value)
print('After', count, sum, sum / count)
```

```
$ python averageloop.py
```

```
Before 0 0
```

```
1 9 9
```

```
2 50 41
```

```
3 62 12
```

```
4 65 3
```

```
5 139 74
```

```
6 154 15
```

```
After 6 154 25.666
```

Середнє арифметичне лише об'єднує шаблони підрахунку та сумування й ділить друге на перше після завершення циклу.

Фільтрування в циклі

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print('Large number',value)
print('After')
```

```
$ python search1.py
Before
Large number 41
Large number 74
After
```

Ми використовуємо інструкцію **if** у циклі, щоби перехопити / відфільтрувати **значення**, які ми шукаємо

Пошук за допомогою логічної змінної

```
found = False
print('Before', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
    print(found, value)
print('After', found)
```

```
$ python search1.py
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

Якщо ми хочемо **шукати й визначити**, чи знайдено потрібне значення, ми використовуємо **змінну**, яка спочатку має значення **Хибне (False)** і змінюється на **Істина (True)**, тільки-но ми знаходимо те, що **шукали**.

Пошук найменшого значення

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

\$ python largest.py

Before -1

9 9

41 41

41 12

41 3

74 74

74 15

After 74

Як ми змінимо код, аби змусити цикл шукати найменше в списку значення?

Пошук найменшого значення

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

Ми перейменували змінну на `smallest_so_far`, а символ `>` змінили на `<`

Пошук найменшого значення

```
smallest_so_far = -1
print('Before', smallest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num < smallest_so_far :
        smallest_so_far = the_num
    print(smallest_so_far, the_num)

print('After', smallest_so_far)
```

\$ python smallbad.py

Before -1

-1 9

-1 41

-1 12

-1 3

-1 74

-1 15

After -1

Ми перейменували змінну на `smallest_so_far`, а символ `>` змінили на `<`

Пошук найменшого значення

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

```
$ python smallest.py
```

```
Before
```

```
9 9
```

```
9 41
```

```
9 12
```

```
3 3
```

```
3 74
```

```
3 15
```

```
After 3
```

Ми все ще маємо змінну, що містить **найменше** значення на зараз. При першому проходженні циклом **найменше** значення є **None**, тому перше взяте значення буде **найменшим**.

Оператори «is» та «is not»

```
smallest = None
print('Before')
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)

print('After', smallest)
```

- У Python є оператор `is`, який можна застосовувати до логічних виразів
- Позначає «такий само, як»
- Схожий, але потужніший за `==`
- `is not` також логічний оператор

Підсумки

- Цикли `while` (невизначені)
- Нескінченні цикли
- Використання `break`
- Використання `continue`
- Константи та змінні `None`
- Цикли `for` (визначені)
- Ітераційні змінні
- Ідіоми циклу
- Найбільше чи найменше

Права власності / застереження



Авторські права на ці слайди з 2010 року належать Чарльзу Северенсу (www.dr-chuck.com) зі Школи інформації Мічиганського університету та застережені ліцензією Creative Commons Attribution 4.0. Будь ласка, збережіть цей фінальний слайд у всіх копіях документа, щоб відповідати вимогам ліцензії щодо посилань на джерела. При повторній публікації матеріалів, якщо щось зміните, додайте ім'я та організацію до переліку співавторів нижче.

Першоджерело: Чарльз Северенс, Школа інформації Мічиганського університету

Переклад: Платформа Prometheus