Розділ 5

Ітерації

5.1 Оновлення змінних

Серед інструкцій присвоєння досить поширеними є ті, що оновлюють змінні, тобто в яких нове значення залежить від попереднього.

```
x = x + 1
```

Наведений вище приклад означає «знайдіть поточне значення x, додайте 1 та оновіть значення x».

Якщо спробувати оновити змінну, якої не існує, виникне помилка, оскільки Python обчислює праву частину перед присвоєнням значення x:

```
>>> x = x + 1
NameError: name 'x' is not defined
```

Перш ніж оновити змінну, її потрібно *ініціалізувати*, зазвичай за допомогою простого присвоєння:

```
>>> x = 0
>>> x = x + 1
```

Оновлення змінної шляхом додавання 1 називається *інкрементом*, а віднімання 1 – *декрементом*.

5.2 Інструкція while

Часто комп'ютери застосовуються для автоматизації завдань, що повторюються. Вони відмінно, без помилок повторюють ідентичні чи схожі завдання, з чим люди справляються набагато гірше. Оскільки досить часто у користувачів виникає потреба у повторенні (ітеруванні) чогось, Python надає кілька вбудованих функцій, які спрощують процес.

Інструкція while – один з різновидів ітерації в Python. Нижче наведений приклад простої програми, яка веде відлік від п’яти до нуля, а потім виголошує «Пуск!».

```
n = 5
while n > 0:
    print(n)
```

```
    n = n - 1
print('Пуск!')
```

Інструкцію while можна прочитати так: «Поки n більше 0, виводь на екран значення n, після чого зменш його на 1. Коли n стане 0, вийди з інструкції while та виведи на екран «Пуск!». Потік виконання інструкції можна записати в три кроки:

1. Обчисліть умову, виведіть значення True (істинно) чи False (хибно).
2. Якщо умова хибна, завершіть роботу інструкції while та перейдіть до виконання наступної інструкції.
3. Якщо умова істинна, виконайте тіло циклу, після цього поверніться до пункту 1.

Такий тип потоку називають *циклом,* оскільки третій крок цієї інструкції повертає назад до початку. Кожне виконання тіла циклу називається *ітерацією.* Можна сказати, що у наведеного вище циклу «п’ять ітерацій», тобто, це означає, що тіло циклу було виконано п’ять разів.

Тіло циклу повинно змінювати значення однієї або кількох змінних так, щоб врешті-решт умова стала хибною і цикл завершився. Змінну, яка змінюється при кожному виконанні циклу і контролює, коли цикл завершується, називають *лічильником циклу* або ж *ітераційною змінною*. Якщо ітераційної змінної немає, цикл повторюватиметься вічно, що призведе до *нескінченного циклу*.

5.3 Нескінченні цикли

Програмістів завжди пробиває на сміх, коли вони бачать інструкції шампунів, у яких вказано «Спінити, змити, повторити», адже це і є той нескінченний цикл, оскільки тут відсутня *ітераційна змінна*, яка вказує скільки разів потрібно виконати цикл.

У випадку зворотного відліку, можна бути впевненими у завершенні циклу, адже відомо, що значення n скінченне і воно з кожною ітерацією зменшується, тож очевидно, що 0 буде досягнуто у будь-якому разі. В інших випадках цикл буде однозначно нескінченним, оскільки в ньому взагалі відсутня ітераційна змінна.

Іноді, тільки після того, як дійдеш до середини тіла циклу, стає зрозуміло, де слід його завершити. У такому випадку можна навмисно написати нескінченний цикл, а потім вийти з нього за допомогою інструкції **break**.

Цілком зрозуміло, що це *нескінченний цикл*, оскільки логічний вираз інструкції while – це просто логічна константа True:

```
n = 10

while True:
    print(n, end=' ')
    n = n - 1
print('Завершено')
```

Ви швидко пожалкуєте, якщо запустите цей код. Самі не встигнете зрозуміти, як ви зуміли зупинити запущений процес в Python чи дотягнутись до кнопки живлення комп’ютера. Ця програма працюватиме вічно, ну, або доти, доки не розрядиться батарея, оскільки логічний вираз у верхній частині циклу завжди істинний завдяки тому, що він є константним значенням True.

Незважаючи на те, що це неробочий нескінченний цикл, його все-таки можна застосувати для побудови ефективніших циклів, якщо уважно додавати в тіло код для явного виходу з циклу за допомогою break, після виконання умови завершення.

Скажімо, вам необхідно прийняти дані від користувача, доки він не введе «завершено». Це можна записати так:

```
while True:
    line = input('> ')
    if line == 'завершено':
        break
    print(line)
print('Завершено')
```

 # Код: http://www.py4e.com/code3/copytildone1.py

Умова циклу – True, а вона завжди істинна, тобто цикл виконуватиметься доти, поки не натрапить на інструкцію break.

Щоразу, підчас виконання циклу, програма виводить користувачеві запит у вигляді кутової дужки. Якщо користувач вводить «завершено», інструкція break виходить з циклу. В іншому випадку програма виводить все, що було введено, і повертається на початок циклу. Розгляньте приклад запуску програми:

```
> привіт
привіт
> закінчити
закінчити
> завершено
Завершено
```

Таким способом написання циклів while користуються досить часто, оскільки є можливість перевірити умову в будь-якому місці циклу (не тільки на початку), а також вказати умову завершення циклу стверджувально (завершити, коли щось відбудеться), а не заперечно (продовжувати виконання, поки не відбудеться щось).

5.4 Завершення ітерацій за допомогою continue

Іноді під час ітерації циклу виникає потреба завершити поточну ітерацію і відразу перейти до наступної. У такому разі, без завершення роботи тіла циклу цієї ітерації, можна скористатися інструкцією continue.

Нижче можна побачити приклад циклу, в якому вхідні дані копіюються доти, доки не буде введено «завершено», однак рядки, що починаються з символу решітки, не виводяться (як на кшталт коментарів Python).

```
while True:
    line = input('> ')
```

```
    if line[0] == '#':
        continue
    if line == 'завершено':
        break
    print(line)
print('Завершено')
```

 # Код: http://www.py4e.com/code3/copytildone2.py

Ось приклад запуску цієї оновленої програми з інструкцією continue.

```
> привіт
привіт
> # не виводь це на екран
> виводь ось це!
виводь ось це!
> завершено
Завершено
```

Усі рядки, окрім того, який починається з символу решітки, виводяться на екран, оскільки інструкція continue завершує поточну ітерацію і переходить до інструкції while, щоб почати наступну ітерацію, пропускаючи інструкцію print.

5.5 Визначені цикли з використанням for

Іноді виникає необхідність перебрати певний перелік елементів *множини*, наприклад, список слів, список рядків у файлі чи список чисел. Якщо у нас є список об'єктів для циклічного відтворення, ми можемо створити *визначений* цикл за допомогою інструкції for. Інструкцію while називають *невизначеним* циклом, оскільки вона просто повторюється, доки якась умова не стане False, тоді як цикл for перебирає визначену множину елементів, тобто виконує стільки ітерацій, скільки є елементів у множині.

Синтаксис циклу for подібний синтаксису циклу while тим, що в ньому є інструкція for і тіло циклу:

```
friends = ['Джозеф', 'Ґленн', 'Саллі']
for friend in friends:
    print('Щасливого Нового Року:', friend)
print('Завершено')
```

Для мови Python, змінна friends – це список[1] з трьох рядків, цикл for перебирає його і виконує тіло по одному разу для кожного з трьох рядків списку, і в результаті виводить такий результат:

```
Щасливого Нового Року: Джозеф
Щасливого Нового Року: Ґленн
Щасливого Нового Року: Саллі
Завершено
```

[1] Детальніше розглянемо список в наступному розділі.

Цей цикл for не так легко прочитати, як попередній while, однак, якщо розглянути перелік друзів, як *множину*, то вийде так: «Виконайте інструкції в тілі циклу for по одному разу для кожного друга з множини friends».

У наведеному циклі for, *for* та *in* – зарезервовані ключові слова Python, а friend та friends – змінні.

```
for friend in friends:
    print('Щасливого Нового Року:', friend)
```

Зокрема, friend – *ітераційна змінна* циклу for. Змінна friend змінюється під час кожної ітерації циклу і контролює, коли цикл for завершиться. *Ітераційна змінна* послідовно перебирає три рядки, що зберігаються у змінній friends.

5.6 Шаблони циклу

Часто цикли for та while використовуються для перегляду списку елементів або вмісту файлу, в якому слід знайти щось на кшталт найбільшого чи найменшого значення даних.

Такі цикли зазвичай будуються за допомогою:

- Ініціалізації однієї або кількох змінних перед початком циклу
- Виконання певних обчислень над кожним елементом у тілі циклу, можливо, змінюючи змінні у тілі циклу
- Перегляду змінних, отриманих в результаті після завершення циклу

Для демонстрації концепцій та побудови цих шаблонів циклу скористаємося списком чисел.

5.6.1 Цикли підрахунку та підсумовування

Наприклад, щоб підрахувати кількість елементів у списку, напишемо такий цикл for:

```
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    count = count + 1

print('Кількість: ', count)
```

Перед початком циклу було встановлено значення змінної count – нуль, а потім написано цикл for, який перебирає список чисел. *Ітераційна змінна* називається itervar, і хоч вона не використовується в циклі, itervar контролює його і змушує тіло циклу виконуватися по одному разу для кожного значення в списку.

У тілі циклу додаємо 1 до поточного значення count для кожного значення у списку. Під час виконання циклу значенням count є кількість значень, які ми «наразі» побачили.

Після завершення циклу, значенням змінної count є загальна кількість елементів. У кінці циклу загальна кількість з'являється, «неначе манна небесна». Ми будуємо цикл таким чином, щоб після його завершення отримати тільки бажаний результат.

Нижче наведено ще один подібний цикл, який обчислює суму наданого набору чисел:

```
total = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    total = total + itervar
print('Разом: ', total)
```

У цьому циклі *ітераційна змінна* використовується. Замість того, щоб просто додавати одиницю до змінної count, як у попередньому циклі, ми додаємо значення поточного елемента (3, 41, 12 тощо) до поточної суми під час кожної ітерації циклу. Якщо звернути увагу на змінну total, то вона містить «поточну суму значень». Отже, перед початком циклу, total дорівнює нулю, оскільки ще не було жодного значення, під час циклу, total – це поточний результат, а в кінці циклу, total – це загальна сума усіх значень у списку.

Під час виконання циклу, total акумулює, тобто накопичує, суму елементів; змінну, що використовується таким чином, іноді називають *акумулятором.*

На практиці, ані цикл підрахунку, ані цикл підсумовування не особливо корисні, оскільки існують вбудовані функції len() і sum(), які обчислюють кількість елементів і загальну суму елементів у списку відповідно.

5.6.2 Цикли максимального та мінімального значень

Щоб знайти найбільше значення у списку чи послідовності, побудуємо такий цикл:

```
largest = None
print('Початкове значення:', largest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if largest is None or itervar > largest :
        largest = itervar
    print('Цикл:', itervar, largest)
print('Найбільше значення:', largest)
```

Після виконання програми буде виведено такий результат:

```
Початкове значення: None
Цикл: 3 3
Цикл: 41 41
Цикл: 12 41
Цикл: 9 41
Цикл: 74 74
Цикл: 15 74
Найбільше значення: 74
```

Змінну largest варто сприймати як «найбільше значення з тих, що ми наразі бачили». Перед циклом встановлюємо largest як константу None. None – це особливе константне значення, яке можна зберігати у змінній, щоб позначити її як «порожню».

Перед початком циклу, найбільшим є None, оскільки поки що не було жодного значення. Під час виконання циклу, якщо найбільше значення є None (if largest is None), беремо перше подане значення як найбільше на цей момент. Зверніть увагу, під час першої ітерації, коли значення itervar дорівнює 3, через те, що найбільшим встановлено None, це значення відразу ж змінюється на 3.

Після першої ітерації значення largest більше не дорівнює None, тому друга частина складеного логічного виразу, яка перевіряє itervar > largest, спрацьовує лише тоді, коли ми бачимо значення, яке більше, ніж «найбільше на цей момент». Коли зустрічаємо нове, «ще більше значення», записуємо його до largest. У вихідних результатах програми можна побачити, що найбільше значення зростає від 3 до 41 і від 41 до 74.

У кінці циклу всі значення було проаналізовано, і змінна largest тепер відображає найбільше значення у списку.

Для обчислення найменшого числа код дуже схожий, лише з однією невеликою зміною:

```
smallest = None
print('Початкове значення:', smallest)
for itervar in [3, 41, 12, 9, 74, 15]:
    if smallest is None or itervar < smallest:
        smallest = itervar
    print('Цикл:', itervar, smallest)
print('Найменше значення:', smallest)
```

Знову-таки, smallest – це «найменше на цей момент значення» до, під час і після виконання циклу. Після завершення циклу smallest містить мінімальне значення зі списку.

Як і з підрахунком та підсумовуванням, у використанні цих циклів немає потреби, адже існують вбудовані функції max() та min().

Нижче наведено просту версію вбудованої функції min():

```
def min(values):
    smallest = None
    for value in values:
        if smallest is None or value < smallest:
            smallest = value
    return smallest
```

У версії коду знаходження найменшого значення за допомогою функції ми видалили всі інструкції print, щоб зробити її еквівалентною функції min, яка вже вбудована в Python.

5.7 Налагодження програми

Під час роботи з великими програмами, більшість часу буде витрачатися на налагодження. Більший код – більша ймовірність зробити помилку та більше місця, де можуть заховатися баги.

Одним способом заощадження часу є «метод налагодження за допомогою бісекції». Наприклад, якщо у програмі 100 рядків, які ви перевіряєте по одному, у вас це займе 100 кроків.

Замість цього спробуйте розділити проблему навпіл. Знайдіть у середині програми або поблизу неї проміжне значення, яке легко перевірити. Додайте інструкцію print (або щось інше, що дозволяє перевірити результат) і запустіть програму.

Якщо під час цієї перевірки видає помилку, це буде свідчити, що проблема виникла в першій частині коду. Якщо все гаразд, то помилка знаходиться в другій половині коду.

Щоразу, під час такої перевірки, ви вдвічі зменшуєте кількість рядків, які потрібно оглянути. Після шести кроків (а це набагато менше 100, погодьтеся), принаймні в теорії, залишиться один або два рядки коду.

Зазвичай, не завжди зрозуміло, де знаходиться «середина програми». Можна не рахувати кількість рядків і шукати точне місце середини, в цьому просто немає сенсу. Натомість поміркуйте про місця в програмі, де можуть знаходитись помилки, і де легко здійснити перевірку. Потім оберіть ділянку, де, на вашу думку, шанси на наявність помилки приблизно однакові – до чи після перевірки.

5.8 Словник

accumulator (акумулятор) Змінна, що використовується в циклі для підсумовування або накопичення результату.
counter (лічильник) Змінна, яка використовується в циклі для підрахунку випадків, коли щось відбулося. Лічильник ініціалізується нулем, а потім збільшується щоразу, коли потрібно щось «полічити».
decrement (декремент) Оновлення, яке зменшує значення змінної.
initialize (ініціалізація) Присвоєння, яке надає початкове значення змінній, що буде оновлюватися.
increment (інкремент) Оновлення, яке збільшує значення змінної (часто на одиницю).
infinite loop (нескінченний цикл) Цикл, в якому ніколи не виконується умова завершення або для якого вона відсутня.
Iteration (ітерація) Повторне виконання набору інструкцій з використанням або функції, яка викликається сама, або циклу.

5.9 Вправи

Вправа 1: Напишіть програму, яка зчитує числа допоки користувач не введе «завершено». Після того, як введено «завершено» виведіть загальну суму, кількість та середнє арифметичне чисел. Якщо введено не число, вкажіть на це за допомогою try та except, виведіть повідомлення про помилку і перейдіть до наступного числа.

```
Введіть число: 4
Введіть число: 5
Введіть число: хибні дані
Помилка введення
Введіть число: 7
Введіть число: завершено
16 3 5.333333333333333
```

Вправа 2: Напишіть ще одну програму, яка робить запит на список чисел, як показано вище, і в кінці замість середнього арифметичного виводить максимальне та мінімальне з них.