3.3. Реалізація циклічних алгоритмів

У реальному житті ми досить часто зустрічаємося з циклами. У комп'ютерних програмах поряд з інструкціями розгалуження (тобто вибором шляху дії) також існують інструкції циклів (повторення дії). Якби інструкцій циклу не існувало, довелося б багато разів вставляти в програму один і той же код поспіль стільки раз, скільки потрібно виконати однакову послідовність дій.

Цикли – це інструкції, які виконують одну й ту ж саму послідовність дій, поки діє задана умова.

Кожен циклічний оператор має тіло циклу – якийсь блок коду, який інтерпретатор буде повторювати поки умова повторення циклу буде залишатися істинною.

В програмуванні розрізняють такі види циклів:

- Цикл з передумовою – виконує дії поки умова є істинною (while).
- Цикл з післяумовою – спочатку виконуються команди, а потім перевіряється умова (do…while).
- Цикл з лічильником – виконується задану кількість разів (for).
- Сумісний цикл – виконує команди для кожного елементу із заданого набору значень (for…which).

В Python присутні лише цикл з передумовою (while) та цикл for, що поєднує в собі два види – цикл з лічильником та сумісний цикл.

Оператор циклу while

Універсальним організатором циклу в мові програмування Python є конструкція *while* [5, 6, 7, 8, 9, 11]. Слово "while" з англійської мови перекладається як "поки" ("поки логічний вираз повертає істину, виконувати певні операції"). Конструкцію *while* на мові Python можна описати наступною схемою:

```
while УМОВА_ПОВТОРЕННЯ_ЦИКЛУ:
    ТІЛО_ЦИКЛУ
```

Дуже схоже на оператор умови – тільки тут використовується інше ключове слово: *while* (англ. "Поки").

Дану конструкцію як правило використовують при повторі уведення даних користувачем, поки не буде отримано коректне значення:

```
correct_choice = False
while not correct_choice:
    choice = input("Введіть число 1 або 2:")
    if choice == "1" or choice == "2":
        correct_choice = True
    else:
        print ("Не правильно введено число, повторіть введеня")
```

Результатом запуску даного коду буде:

```
Введіть число 1 або 2:3
Не правильно введено число, повторіть введеня.
Введіть число 1 або 2:4
Не правильно введено число, повторіть введеня
Введіть число 1 або 2:1
>>>
```

1. Визначили логічну змінну *correct_choice*, присвоївши їй значення *False*.

2. Оператор циклу перевіряє умову *not correct_choice*: заперечення False – істина. Тому починається виконання тіла циклу: виводиться запрошення *"Enter your choice, please (1 or 2):"* і очікується уведення користувача.

3. Після натискання клавіші Enter введене значення порівнюється з рядками *"1"* і *"2"*, і якщо воно дорівнює одній з цих значень, то змінній *correct_choice* присвоюється значення *True*. В іншому випадку програма виводить повідомлення *"Не правильно введено число, повторіть введеня"*.

4. Оператор циклу знову перевіряє умову і якщо вона як і раніше істинна, то тіло циклу повторюється знову, інакше потік виконання переходить до наступного оператору, і інтерпретатор виходить з циклу і виконання програми припиняється або виконуються дії що прописані поза тілом циклу.

Ще один варіант використання оператора циклу – обчислення формул із змінним параметром.

$$\sum_{i=1}^{n} i^3 = 1^3 + \cdots + n^3$$

В даному випадку, параметром, що змінюються є *i*, причому *i* послідовно приймає значення в діапазоні від *1* до *n*.

За допомогою оператора циклу *while* рішення буде виглядати так:

```
n = int(input("Введіть n: "))
sum = 0
i = 1
while i <= n:
    sum += i**3
    i += 1
print ("sum = ", sum)
```

Результатом запуску даного коду буде:

```
Введіть n: 5
sum =  225
```

1. Необхідно ввести *n* – граничне значення *i*.

2. Ініціалізація змінної *sum* – в ній буде зберігатися результат, початкове значення – *0*.

3. Ініціалізація змінної *i* (лічильника *i*) – за умовою, початкове значення – *1*.

4. Починається цикл, який виконується, поки *i* <= *n*.

5. У тілі циклу в змінну *sum* записується сума значення з цієї змінної, отриманої на попередньому кроці, і значення *i*, піднесеної в куб.

6. Лічильнику *i* присвоюється наступне значення.

7. Після завершення циклу виводиться значення *sum* після виконання останнього кроку.

Наступне значення лічильника отримують додаванням до його поточного значення кроку циклу (в даному випадку крок циклу дорівнює *1*). Крок циклу при необхідності може бути від'ємним, і навіть дробовим. Крім того, крок циклу може

змінюватися на кожній ітерації (тобто при кожному повторенні тіла циклу).

Нескінченні цикли

Іноді можна зіткнулися з проблемою нескінченного повторення блоку коду, що виникає, наприклад, через семантичну помилку в програмі:

```
i = 0
while i < 10:
   print (i)
```

Такий цикл буде виконуватися нескінченно, тому що умова *i <10* завжди буде істинною, адже значення змінної *i* не змінюється: така програма буде виводити нескінченну послідовність нулів.

У циклів немає обмеження кількості повторень тіла циклу, тому програма з нескінченним циклом буде працювати безперервно, поки не буде зроблено аварійної зупинки натисканням комбінації клавіш Ctrl+C, не зупинимо відповідний процес засобами операційної системи, або не будуть вичерпані доступні ресурси комп'ютера (наприклад, може бути досягнуто максимально допустимої кількості відкритих файлів), чи не виникне виняток. У таких ситуаціях кажуть, що програма зациклилася.

Знайти зациклення в програмі іноді буває не так-то просто, адже в реальних програмах зазвичай використовується багато циклів, кожен з яких потенційно може виконуватися нескінченно.

Проте, відсутність обмеження на кількість повторів тіла циклу дає нові можливості. Багато програм представляють собою цикл, в тілі якого проводиться відстеження та оброблення дій користувачів або запитів з інших програмних і автоматизованих систем. При цьому такі програми можуть працювати без перебоїв дуже тривалі терміни, іноді роками.

Цикли – це дуже потужний засіб реалізації, але вони вимагають деякої уважності.

Якщо необхідно, щоб цикл виконувався до тих пір, поки щось не станеться, але точно не відомо, коли ця подія трапиться,

можна скористатися нескінченним циклом, що містить оператор ***break***.

```
i = 1

while True:
    if i > 5:
        break                    # Перериваємо цикл
    print (i)
    i += 1
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

1. Виведемо 5 чисел починаючи з *1*. Змінній *i* привласнено початкове значення *1*.

2. Найпростішою умовою для нескінченного циклу є значення True.

3. Виконується перевірка значення змінної *i* зі значенням *5*. Якщо воно співпадає або є більшим, то виконується оператор *break* що переривае виконання циклу.

4. Виводиться на екран значення числа (виконується якщо не спрацював оператор *break).*

5. Збільшується значення змінної *i* на *1*

Іноді потрібно не переривати весь цикл, а лише пропустити по якійсь причині одну ітерацію. ***Continue*** дозволяє перейти до наступної ітерації циклу до завершуючи виконання всіх виразів всередині циклу.

Розглянемо приклад: виводяться на екран цілі числа в діапазоні від 1 до 10, крім чисел 3, 4, 5.

```
i=0

while i<=10:
    i+=1
```

```
    if 3<=i<=5:
        continue
    print(i)
```

Результатом запуску даного коду буде:

```
1
2
6
7
8
9
10
11
```

Альтернативна гілка else

Мова Python дозволяє використовувати розширений варіант оператора циклу:

while УМОВА_ПОВТОРЕННЯ_ЦИКЛУ:
ТІЛО_ЦИКЛУ
else:
АЛЬТЕРНАТИВНА_ГІЛКА_ЦИКЛУ

Поки виконується умова повторення тіла циклу, оператор *while* працює так само, як і в звичайному варіанті, але як тільки умова повторення перестає виконуватися, потік виконання направляється по альтернативній гілці *else* – так само, як в умовному операторі *if*, вона виконається всього один раз.

```
i = 0
while i < 3:
    print (i)
    i += 1
else:
    print ("кінець циклу")
```

Результатом запуску даного коду буде:

```
0
1
```

```
2
кінець циклу
```

Цикл for

В Python ітератори часто використовуються оскільки вони дозволяють проходити структури даних, не знаючи, наскільки ці структури великі і як реалізовані. Можливо пройти по послідовності таким чином:

```
numbers = [1, 2, 3, 4, 5]
i = 0

while i <=len(numbers)-1:
    print(numbers[i])
    i+=1
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

Однак існує більш характерний для Python спосіб вирішення цього завдання:

```
for element in numbers:
    print(element)
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

Списки подібні до *numbers* є одними з ітераційних об’єктів в Python поряд з рядками, кортежами, словниками та деякими іншими елементами.

Ітераційні об’єкти – це ті, вміст яких можна перебрати в циклі. Ітерування по кортежу або списку повертає один елемент за раз. Ітерування по рядку повертає один символ за раз:

```
word = 'cat'
for letter in word:
    print(letter)
```

Результатом запуску даного коду буде:

```
c
a
t
```

Цикл *for* дозволяє перебирати всі елементи вказаної послідовності (список, кортеж, рядок). Цикл спрацює рівно стільки разів, скільки елементів знаходиться в послідовності.

for ЗМІННА in ПОСЛІДОВНІСТЬ:
ТІЛО_ЦИКЛУ

Ім’я змінної в яку на кожному кроці буде розміщено елемент послідовності обирається довільно.

Оператори *break* та *continue* працюють так само як і в циклі *while*. Блок *else* дозволяє перевірити чи виконався цикл *for* повністю якщо ключове слово *break* не було викликане, то буде виконаний блок *else*. Це корисно, якщо потрібно переконатися в тому, що попередній цикл виконався повністю, замість того щоб рано перерватися:

```
numbers = []
for number in numbers:
    print('Цей список має деякий елемент', number)
    break
else:
# Відсутність переривання означає, що елемент відсутній

    print('Елементів немає в списку, чи не так?')
```

Результатом запуску даного коду буде:

```
'Елементів немає в списку, чи не так?'
```

В циклах використання блоку *else* може здатися нелогічним. Можна розглядати цикл як пошук чогось, в такому випадку *else* буде викликатися, якщо нічого не було знайдено.

Ітерування за кількома послідовностям за допомогою функції zip()

Щоб паралельно ітеруватися за кількома послідовностям одночасно можна використати функцію *zip()*:

```
days = ['Monday', 'Tuesday', 'Wednesday']
fruits = ['banana', 'orange']
drinks = ['coffee', 'tea', 'beer']
for day, fruit, drink in zip(days, fruits, drinks):
    print(day, ": drink", drink, "eat", fruit)
```

Результатом запуску даного коду буде:

```
Monday : drink coffee eat banana
Tuesday : drink tea eat orange
```

Функція *zip()* припиняє свою роботу, коли виконується найкоротша послідовність. Один зі списків (fruits) виявився коротшим за інші, тому результат було виведено лише для двох елементів.

Функція *zip()* – ітератор, який повертає кортежі, що складаються з відповідних елементів аргументів-послідовностей.

Генерація числових послідовностей за допомогою функції range()

Функція ***range()*** повертає послідовність чисел в заданому діапазоні без необхідності створювати і зберігати велику структуру даних на зразок списку або кортежу.

Це дозволяє створювати великі діапазони, не використавши всю пам’ять комп’ютера і не обірвавши виконання програми.

range (start, end, step)

Якщо опустити значення *start*, діапазон почнеться з 0. Необхідною є лише значення *end*, що визначає останнє значення, яке буде створено прямо перед зупинкою функції. Значення

step по замовчуванню дорівнює 1, але можна змінити його на -1 [5 - 13].

Якщо просто викликати функцію *range()*, то результату не буде отримано.

```
>>> range(0,10,1)
range(0, 10)
```

Як і *zip ()*, функція *range ()* повертає ітераційний об’єкт, тому потрібно пройти за значеннями за допомогою конструкції *for ... in* або перетворити об’єкт в послідовність (список, кортеж та ін.).

```
for x in range(0, 3):
    print(x)
```

Результатом запуску даного коду буде:

```
0
1
2
```

```
for x in range(2, -1, -1):
    print(x)
```

Результатом запуску даного коду буде:

```
2
1
0
```

Підходи до створення списків

Створити список цілих чисел можна декількома способами.

Можна додавати елементи до списку по одному за раз використовуючи функцію *append()*:

```
number_list = []
print (number_list)

number_list.append(1)
print (number_list)

number_list.append(2)
```

```
number_list.append(3)
number_list.append(4)
number_list.append(5)
print (number_list)
```

Результатом запуску даного коду буде:

```
[]
[1]
[1, 2, 3, 4, 5]
```

Можна використати ітератор та функцію *range()*:

```
number_list = []
for number in range(1, 6):
    number_list.append(number)
print (number_list)
```

Отримаємо:

```
[1, 2, 3, 4, 5]
```

Або ж перетворити в список сам результат роботи функції *range()*:

```
number_list = list(range(1, 6))
print (number_list)
```

Отримаємо:

```
[1, 2, 3, 4, 5]
```

Однак можна створити список за допомогою включення списку.

Спискове включення

Включення – це компактний спосіб створити структуру даних з одного або більше ітераторів. Включення дозволяють вам об’єднувати цикли і умовні перевірки, не використовуючи при цьому громіздкий синтаксис.

Найпростіша форма такого включення виглядає так:

[ВИРАЗ for ЕЛЕМЕНТ in ІТЕРАЦІЙНИЙ_ОБ’ЄКТ]

```
number_list = [number for number in range(1,6)]
print (number_list)
```

Отримаємо:

```
[1, 2, 3, 4, 5]
```

У першому рядку потрібно, щоб перша змінна *number* сформувала значення для списку: слід розмістити результат роботи циклу в змінну *number_list*. Друга змінна *number* є частиною циклу *for*. Щоб показати, що перша змінна *number* є виразом, спробуємо такий варіант:

```
number_list = [number-1 for number in range(1,6)]
print (number_list)
```

Отримаємо:

```
[0, 1, 2, 3, 4]
```

Включення списку може містити умовний вираз:

[ВИРАЗ for ЕЛЕМЕНТ in ІТЕРАЦІЙНИЙ_ОБ'ЄКТ if УМОВА]

Наступне включення створює список, що складається тільки з парних чисел, розташованих в діапазоні від 1 до 5 (вираз *number% 2* має значення *True* для парних чисел і *False* для непарних):

```
number_list = [number for number in range(1,6) if number % 2
== 1]
print (number_list)
```

Отримаємо:

```
[1, 3, 5]
```

Вираз може містити будь-що. Для створення списку з елементів, що будуть вводитися з клавіатури:

```
number_list = [int(input('введіть число: ')) for number in
range(int(input('введіть кількість елементів: ')))]
print (number_list)
```

Отримаємо:

```
введіть кількість елементів: 5
введіть число: 1
введіть число: 0
введіть число: 5
введіть число: 1
введіть число: 3
[1, 0, 5, 1, 3]
```

Вкладені списки

Списки можуть містити елементи різних типів, включаючи інші списки. ***Вкладеними*** називаються списки, які є елементами іншого списку.

Вкладені списки зазвичай використовуються для подання матриць. Змінною *list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]* описано матрицю:

1 2 3
4 5 6
7 8 9

В змінній *list_list* зберігається список з трьома елементами, в кожному з яких зберігається рядок матриці теж у вигляді списків. Витягти рядок можна за допомогою оператора індексування:

```
list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(list_list)

print(list_list [0])
```

Отримаємо:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
```

Щоб звернутися або отримати елемент матриці (вкладеного списку) необхідно вказати два індекси:

```
list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(list_list)
```

```
print(list_list [0])
print(list_list [0][1])
```

Отримаємо:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
2
```

Оператори індексування виконуються зліва на право. Індекс [0] посилається на перший елемент списку list_list, а [1] – на другий елемент внутрішнього списку. Перший індекс визначає номер рядка, а другий – номер елемента (тобто номер стовпчика) в матриці.

Вкладені цикли

Цикли можуть бути вкладені один в одного. При цьому цикли можуть використовувати різні змінні-лічильники.

Найпростіше застосування вкладених операторів циклу – побудова двовимірних таблиць (матриць, масивів), наприклад:

```
i = 1

while i <= 10:
    j = 1
    while j <= 10:
        print (i * j, end="\t")
        j += 1
    print ()
    i += 1
```

Цикл, перевіряючий умову *i<=10,* відповідає за повторення рядків таблиці. У його тілі виконується другий цикл, який виводить добуток *i* і *j* 10 разів поспіль, розділяючи знаками табуляції отримані результати (*end="\t"*).

Функція *print()* без параметра виконує перехід на новий рядок.

В результат виконання даної програми отримаємо

наступну таблицю:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Перший стовпець і перший рядок містять числа від 1 до 10. На перетині *i*-того рядка і *j*-того стовпця знаходиться добуток *i* і *j*. Отримуємо таблицю множення.

Якщо додати ще один цикл, то отримаємо тривимірну таблицю добутків трьох чисел. Таким чином вкладені цикли дозволяють будувати таблиці будь-якої розмірності, що дуже часто застосовується в складних наукових розрахунках.

```
outer = [1, 2, 3]                # Зовнішній цикл
inner = [4, 5, 6]                # Вкладений (внутрішній) цикл
for i in outer:
    for j in inner:
        print ('i=', i, 'j=', j)
```

Результатом запуску даного коду буде:

```
i= 1 j= 4
i= 1 j= 5
i= 1 j= 6
i= 2 j= 4
i= 2 j= 5
i= 2 j= 6
i= 3 j= 4
i= 3 j= 5
i= 3 j= 6
```

Цикл *for* спочатку просувається по елементах зовнішнього циклу (фіксуємо *i=1*), потім переходить до вкладеного циклу

(змінна *j*) і проходимо по всіх елементах вкладеного списку. Далі повертаємося до зовнішнього циклу (фіксуємо значення *i=2*) і знову проходимо по всіх елементах вкладеного списку. Так повторюємо до тих пір, поки не закінчаться елементи в зовнішньому списку:

Даний прийом активно використовується при роботі з вкладеними списками.

Спочатку приклад з одним циклом *for*:

```
lst = [[1, 2, 3], [4, 5, 6]]
for i in lst:
    print (i)
```

Результатом запуску даного коду буде:

```
[1, 2, 3]
[4, 5, 6]
```

У прикладі за допомогою циклу *for* перебираються всі елементи списку, які також є списками.

Якщо необхідно дістатися до елементів вкладених списків, то доведеться використовувати вкладений цикл *for*:

```
lst = [[1, 2, 3], [4, 5, 6]]
for i in lst:                # Цикл за елементами зовнішнього
списку
    print()
    for j in i:              # Цикл по елементах елементів
зовнішнього списку
        print (j, end="")
```

Результатом запуску даного коду буде:

```
123
456
```

Завдання на комп’ютерний практикум

1. Проаналізувати варіант завдання. Скласти програму у відповідності до варіанту та побудувати блок-схему.

Напишіть програму, яка обчислює значення функції (на вхід подається дійсне число):

$Y = \begin{cases} 2x^2 + a \\ (x + 3a)c \end{cases}$	x≤0; a=3,5bx; b=-0,9 x>0; c=1,35x+b$\sqrt{x}$

2. Обчислити значення функції на відрізку [x0; xk] з кроком dx за допомогою циклу з while.
3. Знайти суму n членів ряду, заданого за варіантом. $\sum_{i=1}^{n} \frac{\sqrt{2i}}{\sqrt{i+5} - \sqrt{i}}$
4. Скласти програму яка за введеними з клавіатури значеннями змінних, виконає необхідні розрахунки.

– Знайти суму цифр заданого натурального числа n.
– Дано натуральне число. Підрахувати загальну кількість його дільників.

Запитання для самоконтролю

1. Як описується та виконується оператор розгалуження?
2. Як описується та виконується оператор множинного розгалуження?
3. Що називається логічним виразом?
4. Які 3 можливих варіанти представлення умови в інструкції *if*?
5. Що таке цикл? Навіщо вони потрібні?
6. Як описується та виконується циклічна інструкція while?
7. Як можна організувати нескінченні цикли? Наведіть декілька варіантів і поясніть їх.
8. Як можна вийти з нескінченних циклів?
9. Що відбувається при запуску нескінченного циклу?
10. Чи може оператор циклу не мати тіла? Чому?
11. Для чого служать оператори переривання break та continue? Наведіть приклад.
12. Як працює оператор for?
13. Для організації яких циклів застосовується оператор for?
14. Що таке масиви? Як розташовуються елементи масивів у пам’яті?
15. Як звернутись до першого та останнього елементу масиву?