Програмування

ТЕМА 3. ЦИКЛІЧНІ ПРОГРАМИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА,
КАФЕДРА МАТЕМАТИЧНОЇ ФІЗИКИ. О,В, ОБВІНЦЕВ

Цикл

Циклом називається повторення виконання деякої інструкції *P*

Циклічна програма – це програма яка є ланцюгом команд введення, виведення, присвоєння або тотожної команди, розгалуження а також циклу

3.1 Цикл з умовою продовження

Цикл з умовою продовження

Синтаксис

```
while F:
    P
```

де *F* – умова, *P* - інструкція

Правило циклу з умовою продовження.

- 1. Обчислюється значення F_0 умови *F*.
- 2.1 Якщо F_0 == False, то цикл завершує свою роботу.
- 2.2 Якщо F_0 == True, то виконується інструкція *P* і знову починає виконуватись цикл за цим же правилом.

Приклади циклів з умовою продовження

```
while x > 0:
    x = x - 1


while i < n:
    i = i + 1
    y = y * x


while y > 0:
    x = x -1
```

Скінченність циклів

Останній цикл `while y > 0:`

`x = x -1`,

один раз почавшись, ніколи не закінчиться.

Очевидно, що, якщо інструкція *P* не змінює умову *F*, то цикл буде нескінченним.

Тому **необхідною умовою скінченності циклу** є: інструкція *P* повинна змінювати умову *F*.

Хоарівська трійка

Хоарівська трійка – це трійка

{ *F* } *P* { *G* },

- де *F*, *G* - умови, *P* – інструкція.
- При цьому умова *F* називається передумовою інструкції *P*, а *G* - післяумовою *P*.

Будемо записувати Хоарівську трійку наступним чином

#{F}

P

#{G}

Хоарівська трійка.2

Хоарівська трійка **справджується**, якщо за умови істинності *F* до виконання інструкції *P*, умова *G* буде істинною після виконання *P*.

Приклад Хоарівської трійки, яка справджується:

```
#{x == 1}
x = x + 2
#{x == 3}
```

Властивості циклу з умовою продовження

а) Цикл рівносильний такому розгалуженню

```
while F:      ≡      if F:
    P                    P
                         while F:
                             P
```

Властивості циклу з умовою продовження

b) Справджується трійка

```
#{True}
while F:
    P
#{not F}
```

Приклад: обчислення факторіалу

Обчислити значення $n!$ При заданому n.

$n! = n(n-1)\ldots 1$

$0! = 1$

3.2 Рекурентні співвідношення

Співвідношення 1 порядку

Послідовність $\{a_n\}$ називають заданою **рекурентним співвідношенням 1 порядку** (R_1), якщо

$$\begin{cases} a_0 = c, \\ a_k = f(k, a_{k-1}, p), k = 1,2\ldots \end{cases} (R_1)$$

- Де c – відома константа, f – відома функція, задана у вигляді виразу, p – параметр, що не залежить від номера елемента та елементів послідовності.

Перша теорема про рекурентні співвідношення

Теорема 3.1. (перша теорема про рекурентні співвідношення)

- Нехай послідовність $\{a_n\}$ задана співвідношеннями (R_1).
- Тоді справджується трійка

```
#{t == c and k == 0}
while k < n:
        k = k + 1
        t = f(k,t,p)
#{t == a_n and k == n}
```

Приклад: співвідношення 1 порядку

Обчислити $(-1)^n \dfrac{x^n}{n!}$ при заданих *x* та *n*.

Позначимо $a_n = (-1)^n \dfrac{x^n}{n!}$. Тоді $a_0 = (-1)^0 \dfrac{x^0}{0!} = 1$

$$a_k = (-1)^k \frac{x^k}{k!} \qquad a_{k-1} = (-1)^{k-1} \frac{x^{k-1}}{(k-1)!}$$

$$\frac{a_k}{a_{k-1}} = \frac{(-1)^k x^k (k-1)!}{k!(-1)^{k-1} x^{k-1}} = -\frac{x}{k}$$, звідки $$a_k = -a_{k-1} \frac{x}{k}$$

Приклад співвідношення 1 порядку.2

Маємо рекурентне співвідношення 1 порядку:

$$\begin{cases} a_0 = 1, \\ a_k = -a_{k-1}\dfrac{x}{k}, k = 1,2\ldots \end{cases}$$

Системи співвідношень 1 порядку

Послідовності $\{a_n\}$, $\{b_n\}$ називають заданими **системою рекурентних співвідношень** 1 порядку (R_{11}), якщо

$$\begin{cases} a_0 = c, \\ a_k = f(k, a_{k-1}, p), k = 1,2\ldots \end{cases} \qquad (R_{11})$$
$$\begin{cases} b_0 = d, \\ b_k = g(k, b_{k-1}, a_k, q), k = 1,2\ldots \end{cases}$$

- де c, d – відомі константи, f, g – відомі функції, задані у вигляді виразу, p, q – параметри, що не залежать від номера елемента та елементів послідовностей.

Твердження про системи співвідношень 1 порядку

Твердження 3.1.

- Нехай послідовності $\{a_n\}$, $\{b_n\}$ задані співвідношеннями (R_{11}).
- Тоді справджується трійка

```
#{t == c and u == d and k == 0}
while k < n:
        k = k + 1
        t = f(k,t,p)
        u = g(k,u,t,q)
#{t == a_n and u == b_n and k == n}
```

Приклад системи співвідношень 1 порядку

Обчислити суму $1-x+\frac{x^2}{2}-\ldots+(-1)^n\frac{x^n}{n!}$

при заданих x та n.

- Позначимо n-ий доданок через a_n, а всю суму з $(n+1)$ доданку, - через b_n. Відмітимо, що для послідовності $\{a_n\}$ рекурентне співвідношення вже побудовано.
- Що ж стосується $\{b_n\}$, то маємо $b_0 = 1$, $b_n = b_{n-1} + a_n$.
- Отже, пара послідовностей $\{a_n\}$, $\{b_n\}$ задана системою рекурентних співвідношень 1 порядку.

$$\begin{cases} a_0 = 1, \\ a_k = -a_{k-1}\frac{x}{k}, k = 1,2\ldots \end{cases}$$

$$\begin{cases} b_0 = 1 \\ b_k = b_{k-1} + a_k, k = 1,2\ldots \end{cases}$$

Співвідношення вищих порядків

Якщо елемент послідовності $\{a_n\}$ залежить від декількох попередніх елементів цієї ж послідовності, то кажуть що послідовність $\{a_n\}$ задана **рекурентним співвідношенням порядку вищого, ніж 1**.

Так, для 3 порядку маємо

$$\begin{cases} a_0 = c,\ a_1 = d,\ a_2 = e, \\ a_k = f(k, a_{k-3}, a_{k-2}, a_{k-1}, p), k = 3,4,\ldots \end{cases} (R_3)$$

- де *c, d, e* – відомі константи, *f* – відома функція, задана у вигляді виразу, *p* – параметр, що не залежить від номера елемента та елементів послідовності.

Друга теорема про рекурентні співвідношення

Теорема 3.2. (друга теорема про рекурентні співвідношення)

- Нехай послідовність $\{a_n\}$ задана співвідношенням (R_3).
- Тоді справджується трійка

#{*u* == *c* and v == *d* and w == *e* and k == 2}

while k < *n* + 2:

k = k + 1

t = *f*(k,*u*,*v*,*w*,*p*)

u = *v*

v = *w*

w = *t*

#{*u* == a_n and *v* == a_{n+1} and *w* == a_{n+2} and k == *n* + 2}

Приклад рекурентних співвідношень вищих порядків

Запропонована у Теоремі 3.2 схема обчислень може бути розповсюджена на рекурентні співвідношення довільного порядку, вважаючи, що ми будемо використовувати (m+1) змінну для співвідношення порядку m, а умовою продовження циклу буде k < n + (m-1).

Нехай треба обчислити задане число Фібоначчі. Числа Фібоначчі визначаються рекурентним співвідношенням 2 порядку:

$$\begin{cases} f_0 = 1,\ f_1 = 1, \\ f_k = f_{k-2} + f_{k-1}, k = 2,3,\ldots \end{cases}$$

3.3 Рекурентні обчислення за умовою

Команди break та continue

У циклі while можуть застосовуватись команди

`break`

та

`continue`

Якщо Python зустрічає

`break`

то він перериває виконання циклу.

Якщо Python зустрічає

`continue`

то він пропускає всі команди до кінця циклу та переходить до наступного кроку циклу.

Команди break та continue.2

Команда

`break`

зокрема використовується для реалізації так званих циклів з післяумовою та з виходом.

У цих циклах питання виходу з циклу вирішується не на початку циклу, а в його кінці (або всередині циклу).

Команди break та continue.3

Такий цикл має вигляд:

```
while True:
    P
    if F: break
    Q
```

- При цьому, Q може бути і тотожною командою.

Повний синтаксис while

Повний синтаксис циклу while передбачає також можливість використання else після кінця циклу.

```
while F:
    P
else:
    Q
```

Інструкція *P* може містити, в тому числі, break та continue.

Інструкція Q буде виконуватись у випадку нормального завершення циклу. Якщо ж вихід з циклу здійснюється за допомогою break, то Q не буде виконуватись.

Рекурентні обчислення за умовою

Розглянемо тепер випадок, коли за рекурентним співвідношенням треба обчислити елемент послідовності, що задовольняє певну умову.

Нехай є умова $G(k, x)$, яка залежить від одного числового аргументу. Застосуємо цю умову до членів послідовності $\{a_n\}$, яка задана рекурентним співвідношенням 1 порядку (R_1).

Визначимо через n >= 0 номер першого по порядку члена, який задовольняє цій умові, тобто

$G(0, a_0)$ == False; $G(1, a_1)$ == False; ...; $G(n\text{-}1, a_{n-1})$ == False ; $G(n, a_n)$ == True.

Третя теорема про рекурентні співвідношення

Теорема 3.3. (Рекурентні обчислення за умовою)

При довільній умові $G(k, x)$ для послідовності $\{a_n\}$, яка задана рекурентним співвідношенням 1 порядку (R_1), справджується трійка:

```
#{t == c and k == 0}
while not G(k,t):
        k = k + 1
        t = f(k,t,p)
#{t == a_n and k == n and G(k,t)}
```

Наближене обчислення e^x

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots$$

Відомо, що загальний член цього ряду прямує до 0. Обчислимо наближено e^x як суму цього ряду.

Позначимо загальний член ряду через a_n, а суму, - через b_n.

Будемо вважати точність обчислення задовільною, якщо модуль загального члену ряду менше деякого малого ε, тобто, $|a_n| < \varepsilon$ - це і є умова $G(k, x)$

Наближене обчислення e^x.2

Маємо систему рекурентних співвідношень 1 порядку, в якій нас цікавить b_n такий, що $G(n, a_n)$ == True.

$$\begin{cases} a_0 = 1, \\ a_k = a_{k-1}\dfrac{x}{k}, k = 1,2... \end{cases}$$

$$\begin{cases} b_0 = 1 \\ b_k = b_{k-1} + a_k, k = 1,2... \end{cases}$$

3.4 Цикл по діапазону значень

Цикл по діапазону значень

У Python є також цикл, який виконується задану кількість разів.

При цьому, визначається спеціальна змінна, яка називається лічильником циклу і яка пробігає визначену послідовність значень.

Розглянемо цей цикл спочатку для послідовностей цілих чисел.

Така послідовність повинна бути арифметичною прогресією:

$a_1 = b,\ a_2 = a_1 + d,\ \ldots,\ a_n = a_{n-1} + d,\ \ldots$

Для обмеження кількості повторень циклу встановлюють границю c таким чином, що лічильник пробігає значення всіх елементів послідовності $\{a_n\}$ з напівінтервалу $[b,c)$, при $d > 0$ (напівінтервалу $(c,b]$ при $d < 0$).

Об’єкт range

У Python ця послідовність (прогресія) задається спеціальним об’єктом

```
range(b,c,d)
```

Якщо *d* = 1, то *d* можна опустити і писати

```
range(b,c)
```

Якщо, крім цього, *b* = 0, то *b* також можна опустити і писати

```
range(c)
```

Цикл for

Синтаксис циклу for

```
for i in range(b,c,d):
    P
```

де *b, c, d* – цілі вирази (*d* != 0), *P* – інструкція.

Цикл for.2

Правило виконання циклу for ($d > 0$)

```
for i in range(b,c,d): ≡ if b < c:
    P                        i = b
                             while True:
                                 P
                                 if i + d >= c: break
                                 i = i + d
```

Цикл for.3

При *d* < 0 змінюються знаки двох відношень у правилі виконання циклу for (< на >, а >= на <=):

```
for i in range(b,c,d): ≡  if b > c:
    P                         i = b
                              while True:
                                  P
                                  if i + d <= c: break
                                  i = i + d
```

Цикл for та рекурентні співвідношення

Можна обчислювати елементи послідовностей, заданих рекурентними співвідношеннями, за допомогою циклу for.

Наприклад, для співвідношення 1 порядку (R_1) маємо таку схему обчислень:

#{t == c}

for i in range(1,n+1):

t = f(i,t,p)

#{t == a_n and i == n}

Резюме

Ми розглянули:

1. Поняття циклу та циклічної програми
2. Цикл з умовою продовження, його властивості.
3. Рекурентні співвідношення 1 та вищих порядків, системи рекурентних співвідношень.
4. Правила обчислення елементів послідовностей, заданих рекурентними співвідношеннями.
5. Повний синтаксис циклу за умовою, обчислення границь
6. Цикл по діапазону значень.

Де прочитати

1. Обвінцев О.В. Інформатика та програмування. Курс на основі Python. Матеріали лекцій. – К., Основа, 2017
2. A Byte of Python (Russian) Версия 2.01 Swaroop C H (Translated by Vladimir Smolyar), http://wombat.org.ua/AByteOfPython/AByteofPythonRussian-2.01.pdf
3. Бублик В.В., Личман В.В., Обвінцев О.В.. Інформатика та програмування. Електронний конспект лекцій, 2003 р., http://www.matfiz.univ.kiev.ua/books
4. Марк Лутц, Изучаем Python, 4-е издание, 2010, Символ-Плюс
5. Самоучитель Python. http://pythonworld.ru/samouchitel-python
6. С. Шапошникова. Основы программирования на Python. Версия 2 (2011). http://younglinux.info/pdf
7. Python 3.4.3 documentation