РОЗДІЛ 3. АЛГОРИТМІЧНІ СТРУКТУРИ В МОВІ PYTHON

3.1. Основні алгоритмічні структури

Основними алгоритмічними структурами є: слідування, розгалуження, цикл.

– ***Слідування*** – команди виконуються послідовно одна за іншою.

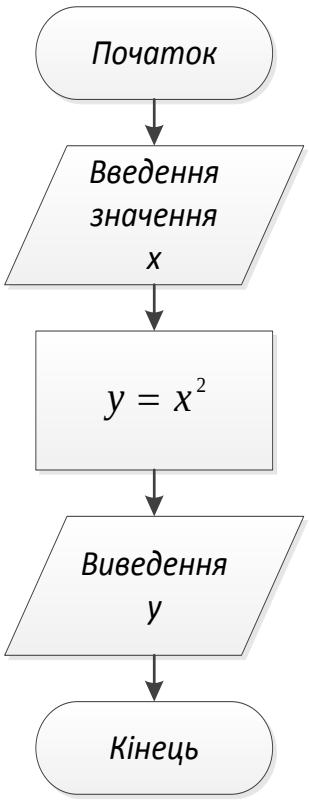


Рис.3.1. Приклад блок-схеми реалізації лінійного алгоритму

– ***Розгалуження*** – алгоритм, що містить хоча б одну умову в результаті перевірки якої може виконуватись розділення на декілька паралельних гілок. Кожна з гілок може містити також розділення, послідовні дії або цикли.

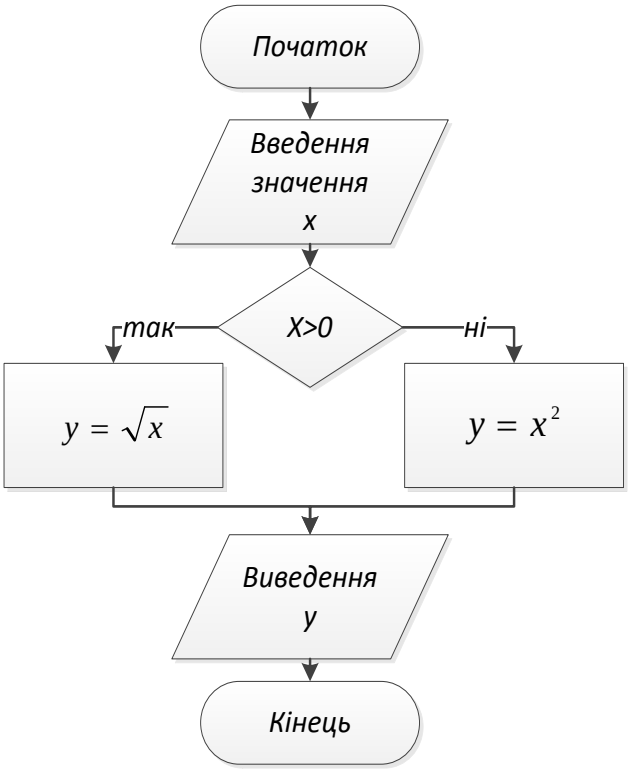
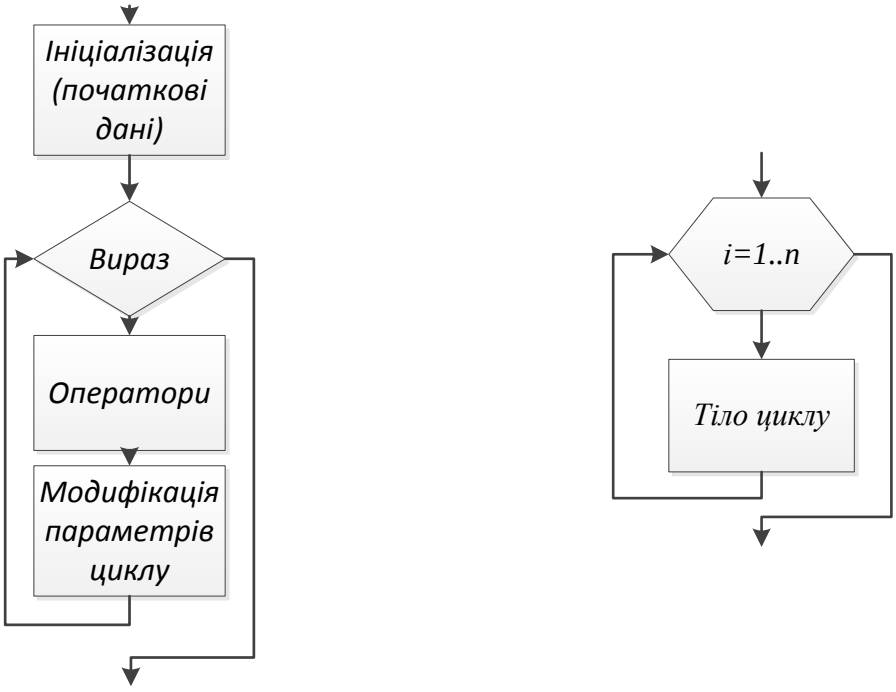


Рис.3.2. Приклад блок-схеми реалізації алгоритму з розгалуженням

– ***Цикл*** – інструкції що виконують одну і ту ж послідовність дій поки діє задана умова.



а б

Рис.3.3. Приклад блок-схеми реалізації циклічного алгоритму (а – цикл з передумовою, б – цикл з лічильником)

3.2. Реалізація алгоритмів з розгалуженням

Хід виконання програми може бути лінійним, тобто таким, коли вирази виконуються, починаючи з першого і закінчуючи останнім, по порядку, не пропускаючи жодного рядка коду. Але частіше буває зовсім не так. При виконанні програмного коду деякі його ділянки можуть бути пропущені.

Припустимо, в реальному житті людина живе за розкладом (можна сказати, розклад – це своєрідний "програмний код", який слід виконати). У її розкладі о 18.00 стоїть похід в басейн. Однак людині надходить інформація, що басейн не працює. Цілком логічно скасувати своє заняття з плавання. Тобто однією з умов відвідування басейну повинно бути його функціонування, інакше повинні виконуватися інші дії.

Схожа нелінійність дій може бути і в комп'ютерній програмі. Частина коду повинна виконуватися лише при певному значенні конкретної умови. Найпростішою в Python для опису розгалужуючої структури, де дії виконуються лише у випадку істинності умови, є така конструкція:

if ЛОГІЧНА_УМОВА:
ПОСЛІДОВНІСТЬ_ВИРАЗІВ

Цю конструкцію на блок-схемі можна зобразити:

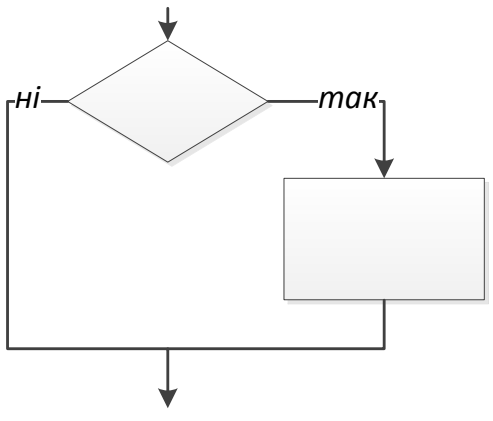


Рис.3.4. Блок-схема реалізації конструкції if

Першим йде ключове слово ***if*** (англ. "Якщо"); за ним – логічний вираз; потім двокрапка, що позначає кінець заголовка оператора, а після неї – будь-яка послідовність виразів або тіло

умовного оператора, яке буде виконуватися в разі, якщо умова в заголовку оператора істинна.

```
x = 2
if x > 0:
    print("x – додатнє")
if x < 0:
    print("x – від’ємне")
```

Результатом запуску даного коду буде:

```
x – додатнє
```

1. Привласнили значення *2* змінній *x*.

2. Зробили умовне порівняння за допомогою операторів *if*, виконуючи різні фрагменти коду в залежності від значень змінної *x*.

3. Викликали функцію *print()*, щоб вивести текст на екран.

Рядки *if* в Python є операторами, які перевіряють, чи є значення виразу (в даному випадку змінна x) рівним *True*.

print() – це вбудована в Python функція для виведення інформації. Вбудовані функції Python – це іменовані фрагменти коду, які виконують певні операції.

Кожен рядок *print()* відокремлений пробілами під відповідною перевіркою.

У більшості мов програмування символи начебто фігурних дужок (*{*і*}*) або ключові слова *begin* і *end* застосовується для того, щоб розбити код на розділи. У цих мовах хорошим тоном є використання відбиття пробілами, щоб зробити програму більш зрозумілою для себе та інших. Існують навіть інструменти, які допоможуть красиво вибудувати код.

Гвідо ван Росум при розробці Python вирішив, що виділення пробілами буде досить, щоб задати структуру програми і уникнути уведення дужок. Python відрізняється від інших мов тим, що пробіли в ньому використовуються для того, щоб задати структуру програми.

Як правило, використовують чотири пробіли для того, щоб виділити кожен підрозділ, хоча можна використовувати будь-яку кількість пробілів, Python чекає, що всередині одного розділу буде застосовуватися однакова кількість пробілів.

Рекомендований стиль – PEP-8 (http://bit.ly/pep-8) – використовувати чотири пробіли. Не рекомендується застосовувати табуляцію або поєднання табуляцій і пробілів – це заважає підраховувати відступи.

З огляду на це, в конструкції *if* код, який виконується при істинності умови, повинен обов'язково мати відступ вправо. Решта коду (основна програма) повинен мати той же відступ, що і слово *if*.

Зустрічається і більш складна форма розгалуження: *if-else*. Якщо умова при інструкції *if* є хибною, то виконується блок коду при інструкції *else*:

```
if ЛОГІЧНА_УМОВА:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_1
else:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_2
```

Цю конструкцію на блок-схемі можна зобразити:

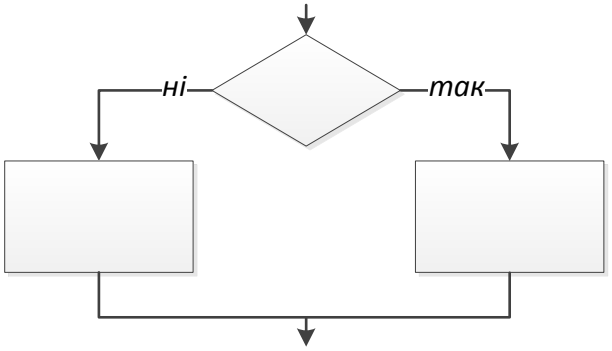


Рис.3.5. Блок-схема реалізації конструкції if-else

Працює ця конструкція наступним чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів. Якщо умова не виконується потік виконання переходить до рядка, який йде після *else*.

```
x = int(input("Введіть x= "))
if x > 0:
    y=x**0.5
else:
    y=x**2
print("y =", y)
```

Отримаємо:

```
Введіть x= 9
y = 3.0
```

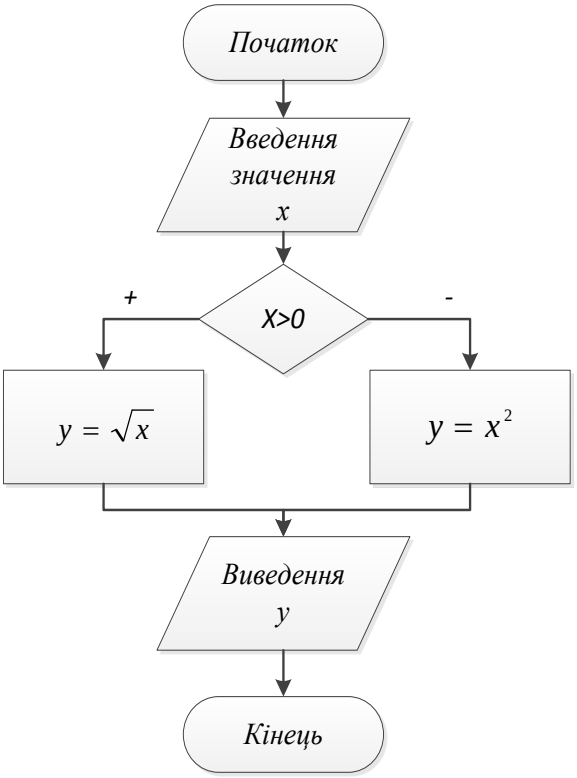


Рис.3.6. Приклад блок-схеми реалізації конструкції if-else

Альтернативні гілки програми

Логіка програми що виконується може бути складнішою, ніж вибір однієї з двох гілок.

Умовний оператор *if* має розширений формат, що дозволяє перевіряти кілька незалежних одна від одної умов і виконувати один з блоків, поставлених у відповідність з цими умовами. У загальному вигляді оператор виглядає так:

```
if ЛОГІЧНА_УМОВА_1:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_1
elif ЛОГІЧНА_УМОВА_2:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_2
elif ЛОГІЧНА_УМОВА_3:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_3
...
else:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_N
```

Цю конструкцію на блок-схемі можна зобразити:

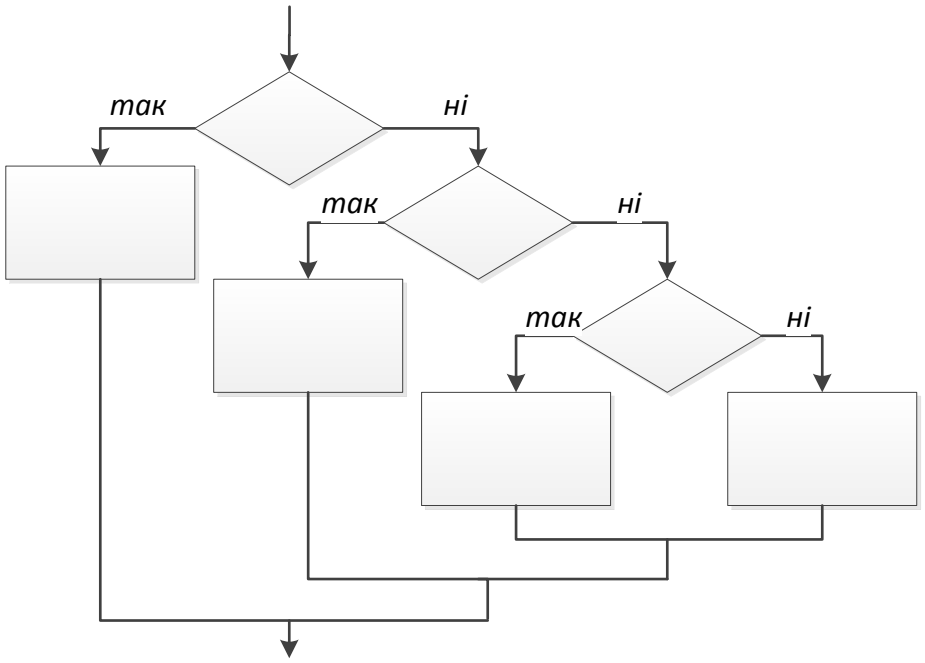


Рис.3.7. Блок-схема реалізації конструкції if-elif-else

Працює ця конструкція наступним чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів. Після цього потік виконання переходить до рядку, який йде після умовного оператора (тобто за послідовністю виразів *N*). Якщо перша умова рівна *False*, то перевіряється друга умова (наступна після *elif*), і в разі його істинності виконується послідовність 2, а потім знову потік виконання переходить до рядка, наступного за оператором умови. Аналогічно перевіряються всі інші умови. До гілки програми *else* потік виконання доходить тільки в тому випадку, якщо не виконується жодна з умов.

Ключове слово *elif* походить від англ. "*Else if*" – "інакше якщо". Тобто умова, яка слідує після нього перевіряється тільки тоді, коли всі попередні умови хибні.

```
if not True:
    print("1")
elif not (1+1==3):
    print("2")
else:
    print("3")
```

Результатом запуску даного коду буде:

```
3
```

Оператор pass

В процесі роботи над програмою слід намагатися після кожної зміни мати працюючу програму, але іноді не завжди відразу відомо, що необхідно виконати якщо умова приймає істинне значення, а що в протилежному випадку. В Python інструкції з розгалуженням, або цикли, або функції з порожнім тілом заборонені, тому в якості тіла використовується "порожній оператор" ***pass***.

Припустимо, що заплановано використання умовного оператора з декількома умовами, але встигнуто написати тільки один з блоків умовного оператора. При цьому постає питання, як її налагодити, якщо програма не виконується через синтаксичну помилку.

```
if not True:
    print("1")
elif not (1+1==3):
elif not (1+1==4):
elif not (1+1==5):
```

Блоки для випадків, коли значення not (1+1==3), not (1+1==4), not (1+1==5), ще не написані, тому програма не виконується через помилку SyntaxError: expected an indented block.

Ключове слово *pass* можна вставити на місце відсутнього блоку:

```
if not True:
    print("1")
elif not (1+1==3):
    pass
elif not (1+1==4):
    pass
elif not (1+1==5):
    pass
```