§2 РОЗГАЛУЖЕНІ ПРОГРАМИ

Розгалуження одна з базових конструкцій алгоритмів. За допомогою розгалужень, програма вибирає яку дію слід виконати, в залежності від значень змінних у момент перевірки.

Основи алгебри висловлювань

Основою розгалужень є алгебра висловлювань. Розглянемо її базові поняття.

Означення

Висловлювання (також використовують терміни логічний або булевий вираз**)** це твердження, якому завжди можна поставити у відповідність тільки одне з двох логічних (булевих) значень: Хибність (**`False`**) або Істина (**`True`**), кожне з яких є запереченням іншого

Приклади істинних висловлювань: «крокодил зелений», «$0 < 1$». Приклади хибних висловлювань: «усі чоловіки обожнюють автомобілі», «$2 + 2 = 5$».

Проте, у програмуванні використовують висловлювання, говорити про істинність або хибність яких можна лише у момент перевірки. Наприклад, висловлювання «$x < 1$» може бути як істинним так і хибним залежно від значення змінної x. Крім того, такі висловлювання можуть бути невизначеними. Такі висловлювання називаються умовами і будуть розглянуті нижче.

Нехай p і q – висловлювання. Над висловлюваннями визначено три базових логічних (булевих) операції:

Диз'юнкція (логічне «або» позначається **`or`**) – це бінарна операція, що діє за правилом: висловлювання p **`or`** q хибне тоді і тільки тоді, коли хибні одночасно висловлювання p і q, та істинне в інших випадках.

Кон'юнкція (логічне «і» позначається **`and`**) – це бінарна операція, що діє за правилом: висловлювання p **`and`** q істинне тоді і тільки тоді, коли істинні одночасно висловлювання p і q, та хибне в інших випадках.

Заперечення (логічне «не» позначається `not`) – це унарна операція, що діє за правилом: висловлювання `not` p є протилежним за змістом до висловлювання p.

Множина B = {`True`, `False`} над елементами якої визначені логічні операції диз'юнкція, кон'юнкція і заперечення називається **булевою алгеброю** або **алгеброю висловлювань**.

Умови

Відношення – це логічна операція, що ставить у відповідність кільком (як правило, двом) величинам булеве значення.

Прикладами відношень можуть бути висловлювання: «3 = 4», «0 < 1», «Микола вищий за Івана».

Операції відношення

У Python існує багато різних операцій відношення, наприклад, перевіри рівності об'єктів, приналежність об'єктів до певної колекції (множини), входження однієї множини до іншої тощо. Очевидно, що можливість застосування кожної операції залежить від типу об'єктів над якими вона проводиться. Наприклад, можна порівнювати дійсні числа на предмет яке з них більше, проте така операція для комплексних чисел не є коректною.

Вивчення операцій відношення є природнім у контексті вивчення типу даних, до елементів якого можуть застосовуватися ці операції. Тому при вивченні кожного нового типу будемо розглядати операції відношення, які можуть застосовуватися до об'єктів цих типів.

Розглянемо операції відношення для числових типів даних. Нехай вирази a і b належать до одного з числових типів – цілого, дійсного або комплексного. Тоді

- `==` (дорівнює) – бінарна операція, що діє за правилом: висловлювання `a == b` істинне тоді і тільки тоді, коли значення виразів `a` і `b` є однаковими;
- `!=` (не дорівнює) – бінарна операція, що діє за правилом: висловлювання `a != b` істинне тоді і тільки тоді, коли значення виразів `a` і `b` є різними;

Наприклад, відношення нижче буде завжди набувати значення **False**

```
5 == 6
```

Вищенаведені відношення можуть застосовуватися не лише для впорядкованих типів, але і для об'єктів будь-яких типів.

Нехай тепер вирази a і b належать до цілого або дійсного типу (тобто впорядкованого). Тоді

- < (менше) – бінарна операція, що діє за правилом: висловлювання `a < b` істинне тоді і тільки тоді, коли значення виразу є меншим за значення виразу;
- <= (менше або дорівнює) бінарна операція, що діє за правилом: висловлювання `a <= b` істинне тоді і тільки тоді, коли значення виразу a є меншим або дорівнює виразу `b`;
- > (більше) – бінарна операція, що діє за правилом: висловлювання `a > b` істинне тоді і тільки тоді, коли значення виразу a є більшим за значення виразу b;
- >= (більше або дорівнює) бінарна операція, що діє за правилом: висловлювання `a >= b` істинне тоді і тільки тоді, коли значення виразу `a` є більшим або дорівнює виразу `b`.

Приклади відношень

```
5 >= 6
x < 0
```

Операції відношення знайомі нам з математики. Проте в математиці використовувалися інші позначення, наприклад «не дорівнює» позначається «≠». Те саме стосується і логічних операцій, наприклад кон'юнкція у курсі математична логіка позначається як «&».

У цьому курсі будемо користуватися здебільшого позначеннями для арифметичних, логічних операцій, операцій відношення та інших операцій тими позначеннями, які використовуються у мові програмування Python.

Пріоритет логічних операцій та операцій відношення

Доповнимо таблицю **1.4** пріоритету операцій операціями відношення та логічними операціями.

Таблиця 2.1. Пріоритет операцій від найвищого до найнижчого

`**`
`*, /, //, %`
`+, -`
`==, !=, >, <, >=, <=`
`not`
`and`
`or`

Зауваження. Нагадаємо, що зміну пріоритету операцій можна здійснювати за допомогою дужок. Також радимо використовувати дужки у випадку, якщо є певні сумніви у тому у якому порядку виконаються операції у логічному виразі або для того, щоб зробити вираз легшим для сприйняття.

Умова – це логічний (булевий) вираз, утворений за допомогою відношень та логічних операцій, про істинність або хибність якого можна стверджувати лише у конкретний момент виконання програми.

Наприклад, умова того, що точка площини з координатами x та y належить першого координатному квадранту буде виглядати таким чином

```
x >= 0 and y >= 0
```

На відміну від звичайних висловлювань, які згідно з означенням завжди є або істинним або хибним, умова може бути невизначеною. Наприклад, вищенаведена умова невизначена, якщо не задані значення змінних `x` та `y`.

Приклад невизначеної умови при заданих значеннях змінних: умова 1/x > y невизначена при x = 0.

Тому, записуючи умови, необхідно враховувати три випадки: істинність, хибність та невизначеність.

У подальшому, якщо для деякого фіксованого стану програми умова набуває істинного значення (тобто значення `True`), то будемо казати, що **умова виконується**, а якщо хибного (`False`) – **умова не виконується**.

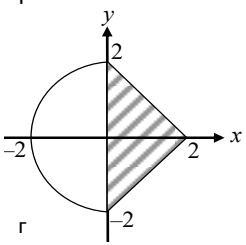
Приклад 2.1. Записати умову можливості існування трикутника із заданими сторонами a, b, c.

Розв'язок. Якщо згадати, що в будь-якому трикутнику кожна сторона менша від суми двох інших сторін, можливість існування трикутника рівносильне виконанню умови:

```
(a + b > c) and (a + c > b) and (b + c > a)
```

Звертаємо увагу на те, що дужки у цьому виразі є необов'язковими. Дійсно, арифметична операція «+» має вищий пріоритет ніж операція «>», а операція відношення «>» має вищий пріоритет ніж кон'юнкція. Проте, для того, щоб зробити умову читабельною у цьому виразі використовуються дужки.

Приклад 2.2. Записати умову приналежності точки площини з координатами (x, y) до зафарбованої області



Розв'язок. Логічні операції у геометричному контексті тісно пов'язані з операціями над множинами. Так, наприклад, якщо умови F_1 та F_2 визначають множини M_1 та M_2, то:

умова F_1 **and** F_2 визначає перетин $M_1 \cap M_2$ цих множин,

умова F_1 **or** F_2 визначає об'єднання $M_1 \cup M_2$,

умова **not** F_1 визначає доповнення до множини M_1.

Визначимо заштриховану область через операції з множинами. Заштрихована область буде об'єднанням заштрихованих підобластей з лівої $(x \le 0)$ та правої $(x > 0)$ півплощини. Отже

$$(\{x \le 0\} \cap \{x^2 + y^2 \le 4\}) \cup (\{x \ge 0\} \cap \{|x| + |y| \le 2\})$$

Звідки очевидним чином, замінивши операції над множинами логічними операціями, отримуємо умову приналежності точки до заштрихованої області.

```
((x <= 0) and (x ** 2 + y ** 2 <= 4)) or
    ((x >= 0) and (abs(x) + abs(y) < 2))
```

Логічний тип даних у Python

Реалізація алгебри висловлювань у Python представлена логічним типом даних `bool`. Змінні цього типу можуть набувати лише двох значень: **`True`**, **`False`**.

Результатом логічних операцій та умов завжди є булеві літерали **`True`** або **`False`**.

При необхідності Python автоматично зводить вираз до значення типу `bool` таким чином:

- Будь-яке число, що не дорівнює **`0`**, або непорожній об'єкт – це **`True`**.
- Число **`0`**, порожній об'єкт або значення **`None`** – це **`False`**.

Приклад 2.3. Точка площини задана декартовими координатами (x, y). Перевірити чи належить вона третьому координатному квадранту.

Розв'язок. Результатом виконання програми слід вважати значення логічного типу. При цьому будемо вважати, що значення **`True`** інтерпретується як відповідь «так» або «правда», а значення **`False`** – як відповідь «ні» або «неправда». Отже, умова буде мати вигляд:

```
x < 0 and y < 0
```

Тоді програма матиме вигляд:

```
x = float(input("x = "))
y = float(input("y = "))
check = x < 0 and y < 0  # Обчислюємо значення умови
print("Точка належить до 3-го квадранту це ", check)
```

Розгалуження

Розгалуження – це конструкція мови програмування, що забезпечує виконання визначеної команди (набору команд) у випадку виконання деякої умови або виконання однієї з кількох команд (набору команд), в залежності від значення деякої умови.

Реалізація розгалуження у Python представлена кількома керуючими конструкціями:

- умовним оператором;
- каскадним розгалуженням;
- тернарним умовним оператором.

Умовний оператор

Умовний оператор дозволяє виконувати певний набір інструкцій у випадку виконання деякої умови і інший набір інструкцій у випадку її невиконання.

Умовний оператор має такий синтаксис:

```
if condition:
    state1
else:
    state2
```

де `condition` називається умовою розгалуження, `state1` та `state2` – альтернативами.

Правила роботи умовного оператора таке: інтерпретатор обчислює значення умови `condition` і далі

- якщо умова `condition` виконується, то виконується інструкція (набір інструкцій) `state1`.
- у іншому разі, якщо умова `condition` не виконується, то виконується набір команд `state2`

Інструкції `state1` і `state2` називаються **гілками** умовного оператора.

Команди `state1` і `state2` є вкладеними інструкціями для умовного оператора. Тому, до них має застосовуватися форматування як для вкладених інструкцій – **відступ розміром 4 пробіли відносно основної інструкції.**

Приклад 2.4. Знайти модуль дійсного числа x введеного з клавіатури.

Розв'язок. Для розв'язання задачі просто скористаємося означенням модуля дійсного числа:

$$|x| = \begin{cases} x, & x \geq 0, \\ -x, & x < 0. \end{cases}$$

Таким чином програма буде мати такий вигляд:

```
x = float(input("x = "))     # Введення дійсного x
if x >= 0:
    abs_x = x
else:
    abs_x = -x
print("|", x, "|= ", abs_x) # Виведення результату
```

В умовному операторі блок **else** є необов'язковим. Тому, у ситуації, коли виконання певної інструкції (набору інструкцій) необхідно провести лише у випадку виконання деякої умови, то блок **else** опускають. При цьому такий умовний оператор називають **умовним оператором з однією гілкою**:

```
if condition:
    state
```

Виконання умовного оператора з однією гілкою буде полягати у такому: інструкція `state` виконується лише у випадку виконання умови `condition`. У зв'язку з цим, умовний оператор з однією гілкою інколи називають **захищеною інструкцією**.

Приклад 2.5. Знайти модуль дійсного числа x введеного з клавіатури.

Розв'язок. Скористаємося захищеною інструкцією у такому вигляді: якщо введене з клавіатури число є від'ємним, то поміняємо його знак на протилежний

```
x = float(input("x = "))     # Введення x
abs_x = x
if x < 0:                    # Якщо x – від'ємний
    abs_x = -x               # Міняємо знак x
print("|", x, "|= ", abs_x) # Виведення результату
```

Приклад 2.6. Скласти програму для знаходження найбільшого з трьох значень, введених з клавіатури.

Розв'язок. Нехай $\max(\ldots)$ визначає найбільше серед чисел, що стоять у дужках. Тоді легко бачити, що $\max(x,y,z) = \max(max(x,y),z)$. Тому у програмі спочатку визначимо більше зі значень x та y, а вже потім – найбільше серед трьох чисел.

```
x = float(input("x = "))
y = float(input("y = "))
z = float(input("x = "))
# Визначаємо більше серед чисел x та y
if x < y:
    max = y
else:
    max = x
# Визначаємо більше серед чисел max(x,y) та z.
if max < z:
    max = z
print("max(", x, y, z, ")= ", max)
```

Каскадне розгалуження

Крім звичайного умовного оператора у Python є ще оператор **каскадного (множинного) розгалуження**. Каскадне розгалуження дозволяє виконати певний набір інструкцій у випадку виконання однієї серед кількох умов.

Синтаксис каскадного розгалуження:

```
if condition1:
    state1
elif condition2:
    state2
…
elif condition_N:
    state_N
else:
    state_else
```

Каскадне розгалуження виконується таким чином: інтерпретатор обчислює умову `condition1` і якщо вона виконується то інтерпретатор виконує блок інструкцій `state1`. Якщо умова `condition1` не виконується, то інтерпретатор переходить до блоку **`elif`** і обчислює умову `condition2`. Якщо умова `condition2` виконується то виконує блок інструкцій `state2`, а якщо ні, то переходить до наступного блоку **`elif`** і т.д. Якщо жодна з умов `condition1`, …, `condition_N` не є істиною, то інтерпретатор виконує блок інструкцій `state_else`.

З точки зору Python (та й будь-якої іншої мови програмування), каскадне розгалуження є послідовністю вкладених розгалужень. Саме це і породило назву для цього оператора. Отже, вищенаведене каскадне розгалуження рівносильне такому:

```
if condition1:
    state1
else:
    if condition2:
        state2
        ……
                else:
                    if condition_N:
                        state_N
                    else:
                         state_else
```

Приклад 2.7. Для введеної з клавіатури точки x знайти значення функції

$$f(x) = \begin{cases} -x^2 + 1, & x < -1, \\ 0, & |x| \le 1, \\ x^2 - 1, & x > 1. \end{cases}$$

Розв'язок. Скористаємося каскадним розгалуженням:

```
x = float(input("x = "))
if x < -1:
    f = -x ** 2 + 1
elif abs(x) <= 1:
    f = 0
else:
    f = x ** 2 - 1
print("f(", x, ")= ", f)
```

Тернарний умовний оператор

Тернарний умовний оператор (умовний вираз), це операція, що повертає одне з двох значень залежно від істинності чи хибності заданої умови. Його синтаксис такий:

```
expresion1 if condition else expresion2
```

де `condition` – умова, `expresion1` та `expresion2` – деякі вирази (значення яких можуть бути присвоєні змінній). Оператор повертає значення виразу `expresion1`, якщо умова `condition` виконується і `expresion2` у іншому разі.

Отже, згідно з визначенням тернарного оператора, інструкція

```
result = expresion1 if condition else expresion2
```

рівносильна такому розгалуженню

```
if condition:
    result = expresion1
else:
    result = expresion2
```

Використання тернарного умовного оператора дозволяє компактніше записувати програмний код

Приклад 2.8. Знайти модуль дійсного числа x введеного з клавіатури.

Розв'язок. Скористаємося тернарним умовним оператором:

```
x = float(input("x = "))
```

```
abs_x = x if x >= 0 else -x
print("|", x, "|= ", abs_x)
```

Приклад 2.9. Знайти більше з двох значень, введених із клавіатури. *Розв'язок*.

```
x = float(input("x = "))
y = float(input("y = "))
max2 = x if x > y else y
print("max(", x, y, ")= ", max2)
```

Задачі для аудиторної роботи

2.1. Записати умови, що істинні тоді й тільки тоді, коли:
a) натуральне число n – парне;
b) остання цифра числа $n - 0$;
c) сума першої і другої цифри двозначного натурального числа n – двозначне число.

2.2. Точка площини задана декартовими координатами (x, y). Перевірити, чи належить вона кільцю, з центром у початку координат, внутрішнім радіусом 1 і зовнішнім 2.

2.3. Дано тризначне число. Перевірити чи
a) містить воно цифру 2;
b) складається лише з парних чисел;
c) сума його цифр дорівнює 18.

2.4. Для заданого x обчислити значення функцій

$$f(x) = \begin{cases} 0, & x \le 0, \\ x^2, & 0 < x \le 1, \\ x^4, & x > 1. \end{cases}$$

Задачі для самостійної роботи

2.5. Перевірити впорядкованість змінних a, b, c:
a) за зростанням їхніх значень;
b) за спаданням їхніх значень;
c) за зростанням чи спаданням їхніх значень.

2.6. Записати умови, що істинні тоді й тільки тоді, коли:
a) натуральне число n – непарне;
b) остання цифра натурального числа n є m;
c) ціле число n кратне натуральному числу m;
d) натуральні числа n і k одночасно кратні натуральному числу m;
e) сума цифр тризначного числа належить проміжку (a, b);