Оператор raise дозволяє програмісту згенерувати вказаний виняток. В єдиному аргументі raise вказується виняток, який буде викликано, наприклад:

```
>>> raise ZeroDivisionError('Ділення на нуль.')
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    raise ZeroDivisionError('Ділення на нуль')
ZeroDivisionError: Ділення на нуль
```

5. Організація розгалужень в програмах

5.1. Логічні вирази і логічний тип даних

Досить часто в реальному житті зустрічаються твердження, з якими ми погоджуємося чи ні. Наприклад, якщо вам скажуть, що сума чисел 2 та 3 більше 4, ви погодитесь і скажете: "Так, це правда". Якщо ж хтось буде стверджувати, що сума чисел 2 та 3 менше 4, то ви сприймете це твердження як хибне.

Подібні твердження допускають лише дві можливих відповіді - або "так", коли твердження оцінюється як істинне (правдиве), або "ні", коли твердження оцінюється як хибне (помилкове). Такі твердження ще називають логічними виразами або логічними твердженнями. Логічний вираз в програмуванні – це конструкція мови програмування, результатом обчислення якої є «істина» або «хиба».

Логічним (булевим) типом даних в мові Python є тип bool, що може набувати одного з двох значень: True (істина) або False (хиба). Проте в мові Python істинним або хибним може бути не лише логічний вираз, але і об’єкт.

- Число не рівне нулю, або непорожній об’єкт інтерпретується як істина.
- Нуль, порожні об’єкти і спеціальний об’єкт None інтерпретуються як хиба.

5.2. Оператори відношень (порівнянь)

В Python для порівняння об’єктів (змінних різних типів) є наступні операції порівняння:

- > – більше;

- < – менше;
- >= – більше або рівне (не менше);
- <= – менше або рівне (не більше);
- == – дорівнює (рівне);
- != – не дорівнює (не рівне).

Результатом операції порівняння є змінна логічного типу bool.

```
>>> 6>5
True
>>> 7<1
False
>>> 7==7
True
>>> 7!=7
False
```

5.3. Умовний оператор if-else (if-elif-else)

Всі раніше розглянуті програми мали лінійну структуру, тобто всі інструкції виконувалися послідовно одна за одною, і кожна записана інструкція обов'язково виконувалась. За необхідності змінити порядок виконання операторів програми в залежності від виконання певних умов, тобто здійснити розгалуження процесу обчислень, використовують умовний оператор. [7]

Наприклад, необхідно для заданого числа x визначити: воно додатне, якщо $x \geq 0$, чи від'ємне в іншому випадку. Структура програми вже не може бути лінійною, оскільки залежно від значення x ($x \geq 0$ або $x < 0$) має бути виведене одне чи інше повідомлення.

Для розв'язання цієї задачі можна скористатися умовним оператором if-else, який ще називається оператором розгалуження. Використовуючи вказаний оператор, можна організувати виконання тих чи інших операторів в залежності від деякого логічного виразу (умови).

Синтаксис оператора if-else:

```
if Логічний_вираз:
    Блок_інструкцій_1
```

```
[else:
    Блок_інструкцій_2]
```

Під час виконання оператора if-else обчислюється значення логічного виразу. Якщо значення логічного виразу істинне (True), то виконується `блок_інструкцій_1` (вирази, що є вкладеними в if). Якщо значення логічного виразу хибне (False), виконується `блок_інструкцій_2` (вирази, що є вкладеними в else).

Умовний оператор може бути неповним, якщо в ньому відсутня гілка else, тобто відсутнє службове слово else з відповідним йому блоком інструкцій.

Як можна бачити, запис службового слова if з логічним виразом та службового слова else і відповідних їм блоків інструкцій оформлюються в коді, як основна інструкція та вкладений блок інструкцій. Тобто всі інструкції, які відносяться до одного вкладеного блоку, повинні мати рівну величину відступу, тобто однакове число пропусків на початку рядка. Рекомендується використовувати відступ в 4 пропуски і не рекомендується використовувати в якості відступу символ табуляції.

Приклад. Написати програму для визначення, чи є задане число додатним чи не є додатним:

```
x = int(input("Введіть число: "))
if x > 0:
    print("Число додатне. ")
else:
    print("Число не є додатне. ")
```

Розширивши умову задачі необхідності визначення, чи є число додатним, чи є число від'ємним чи число є нулем (число 0 не є додатним і не є від'ємним). Виникне необхідність доповнити програму додатковою можливістю перевірки числа на рівність нулю і виведенням відповідного повідомлення.

Це можна зробити декількома способами.

Спосіб 1. Використання окремих перевірок.

```
x = int(input("Введіть число: "))
if x > 0:
    print("Число додатне. ")
```

```
if x < 0:
    print("Число від'ємне. ")
if x == 0:
    print("Число нуль. ")
```

У цій програмі відбувається перевірка всіх можливих випадків (більше, менше, рівне) з виведенням відповідних повідомлень. Варто відмітити, що для перевірки 3-х взаємовиключних умов можна скористатися методом виключення третього (якщо не перше і не друге, то третє) і використати лише дві перевірки.

Перепишемо програму.

Спосіб 2. Використання вкладених перевірок.

```
x = int(input("Введіть число: "))
if x > 0:
    print("Число додатне.")
else:
    if x < 0:
        print("Число від'ємне.")
    else:
        print("Число нуль.")
```

З наведеного коду програми можна бачити, що вона містить два оператори if-else, один з яких вкладений в інший. Тобто оператор if-else для умови x < 0 вкладений (внутрішній) в зовнішній оператор if-else для умови x > 0.

Програма буде працювати таким чином:

- Спочатку виконується зовнішній оператор, в якому перевіряється умова x > 0.
- Якщо умова x > 0 істинна (значення змінної x більше нуля), то виводиться повідомлення, що число додатне.
- Якщо умова x > 0 хибна (значення змінної x не більше нуля, тобто нуль або від'ємне), переходимо до виконання вкладеного оператора if- else. Перевіряється умова x < 0.
- Якщо умова x < 0 істинна (значення змінної x менше нуля), то виводиться повідомлення, що число від'ємне.

- Якщо умова x < 0 хибна (значення змінної x рівне нулю, оскільки не виконалась умова x > 0, і не виконалася умова x < 0), то виводиться повідомлення, що число нуль.

Проте в мові Python передбачений спрощений запис для виконання таких вкладених перевірок. Для їх реалізації використовується оператор if-elif-else, яка має вигляд:

```
if Логічний_вираз_1:
    Блок_інструкцій_1
elif Логічний_вираз_2:
    Блок_інструкцій_2
elif Логічний_вираз_3:
    Блок_інструкцій_3
…
[else:
    Блок_інструкцій_N]
```

Оператор працює наступним чином. Обчислюється значення логічного виразу 1. Якщо значення `логічного_виразу_1` істинне, то виконується `блок_інструкцій_1`. Якщо ж значення `логічного_виразу_1` хибне, то відбувається перехід до обчислення значення логічного виразу 2. Якщо значення `логічного_виразу_2` істинне, то виконується `блок_інструкцій_2` і так далі. `Блок_інструкцій_N` буде виконаний в тому випадку, якщо жодний з логічних виразів не був істинним.

Враховуючи оператор if-elif-else, перепишемо нашу програму.

Спосіб 3. Використання оператора if-elif-else.

```
x = int(input("Введіть число: "))
if x > 0:
    print("Число додатне.")
elif x < 0:
    print("Число від'ємне.")
else:
    print("Число нуль.")
```

Оператор if-elif-else може використовуватися для заміни оператора switch-case або case в інших мовах програмування.

5.4. Тримісний оператор if/else

Як і в деяких інших мовах програмування, в мові Python передбачений тримісний оператор if/else, який в окремих випадках більш доцільно використовувати замість оператора if-else. Незважаючи на те, що сфера його застосування більш вузька.

Наприклад, нам необхідно в залежності від деякої умови надати одне чи інше значення змінній (наприклад, змінній a присвоюється значення виразів Y або Z в залежності від істинності умови X). З використанням оператора if-else це запишеться так:

```
if X:
    a = Y
else:
    a = Z
```

Використовуючи тримісний оператор if/else, це можна записати так:

```
a = Y if X else Z
```

5.5. Логічні оператори

Інколи є необхідність будувати більш складні логічні вирази, які будуть містити декілька простих логічних тверджень, та між якими необхідно виконати логічні оператори: І, АБО, НІ (and, or, not).

Операндами операторів and, or та not можуть бути як логічні вирази (результат яких має логічний тип), так і вирази, результат яких не є логічного типу (число, рядок, список і т.д.). Тому в загальному можна сказати, що операндами операторів and, or та not є об'єкти.

Логічний оператор not (НЕ)

Логічний оператор not також називають запереченням.

Використання оператора: not X.

Результатом застосування логічного оператора not є значення логічного типу, яке є запереченням операнда.

Якщо операнд істинний (True, будь-яке число не рівне нулю, або не порожній об'єкт), то за оператором not буде повернуто – False. Якщо операнд

хибний (False, нуль, порожній об'єкт або спеціальний об'єкт None), то за оператором not буде повернуто – True.

Логічний оператор and (І)

Логічний оператор and також називають кон'юнкцією або логічним множенням.

Використання оператора: X1 and X2[and X3 ...].

Результатом застосування логічного оператора and є об'єкт.

При обчисленні оператора and операнди обчислюються зліва направо. Як тільки знайдено перший об'єкт, що має хибне значення (інтерпретується як хибне), він вважається результатом обчислення оператора and, і подальше обчислення завершується. Таке раннє завершення обчислення називається обчисленням за короткою схемою. Якщо серед операндів не знайдено об'єкта, що має хибне значення, то повертається крайній правий об'єкт.

```
>>>0 and 3 #повертає перший хибний об'єкт
0
>>>5 and 4  #повертає крайній правий об'єкт
4
```

Якщо операндами оператора and є логічні вирази, то процес обчислення більш спрощений і його можна описати так. Логічний оператор and повертає істину, якщо всі операнди будуть істинними, якщо ж принаймні один з операндів буде хибним, то логічний оператор and поверне хибу.

```
>>> 2>4 and 45>3
False
```

Логічний оператор or (АБО)

Логічний оператор or також називають диз'юнкція або логічне додавання. Використання оператора: X1 or X2[or X3 ...].

Результатом застосування логічного оператора or є об'єкт.

При обчисленні оператора or операнди обчислюються зліва направо. Як тільки знайдено перший об'єкт, що має істинне значення (інтерпретується як істинне), він вважається результатом обчислення оператора or і подальше обчислення завершується. Таке раннє завершення обчислення називається обчисленням за короткою схемою. Якщо серед операндів не знайдено об'єкта, що має істинне значення, то повертається крайній правий об'єкт.

```
>>>2 or 3  # повертає перший істинний об'єкт
2
>>> None or 0  # повертає крайній правий об'єкт
0
```

Якщо операндами оператора or є логічні вирази, то процес обчислення можна описати так. Логічний оператор or повертає істину, якщо принаймні один операнд буде істинним, якщо ж всі операнди будуть хибними, то логічний оператор or поверне хибу.

Розв'яжемо невелику задачу. Обчислити значення виразу 1/x без використання умовного оператора та обробки винятків. Особливістю даної задачі є те, що у випадку, коли x буде рівне 0, виконання ділення 1/x призведе до виникнення помилки. Використовуючи особливості логічних операторів, для розв'язання даної задачі можна записати наступний вираз: `x and 1/x`.

```
>>> x=1
>>> x and 1/x
1.0
>>> x=0
>>> x and 1/x
0
```

Логічні вирази можна комбінувати:

```
>>> 1+3 > 7  # пріоритет операції «+» вище, чим в «>»
False
```

Для більшої зрозумілості можуть використовуватися дужки.

```
>>> (1+3) > 7
False
```

Поміркуйте, що буде виведене в результаті виконання виразу, і чому буде отриманим саме такий результат:

```
>>> 1+(3>7)
1
```

В Python можна перевіряти приналежність інтервалу:

```
>>> x=0
>>> -5<x<10  # еквівалентне: x > -5 and x<10
True
```

5.6. Приклади розв'язування задач

Приклад. Написати програму, за якою будуть знайдені корені квадратного рівняння, заданого своїми коефіцієнтами.

```
import math
a=float(input('Введіть коефіцієнти рівняння a,b,c:'))
b=float(input(''))
c=float(input(''))
d=b**2-4*a*c
if d>=0:
     x1=(-b-math.sqrt(d))/(2*a)
     x2=(-b+math.sqrt(d))/(2*a)
     print('x1={:.4}, x2={:.4}'.format(x1,x2))
else:
     print('Дійсних коренів немає')
```

Приклад. Написати програму, за якою буде визначитися, чи є заданий рік високосним. Відповідно з Григоріанським календарем, рік є високосним, якщо його номер кратний 4, але не кратний 100, а також, якщо він кратний 400.

```
year=int(input('Введіть рік: '))
if (year % 4 == 0 and not year % 100 == 0) or
          (year % 400 == 0):
     print('Рік високосний.')
else:
     print('Рік не високосний.')
```

Приклад. Написати програму, за якою буде визначено, чи є дві клітинки, задані користувачем, одного кольору чи ні. Для задання клітинки шахової дошки необхідно вказати два числа від 1 до 8, які будуть визначати номер рядка та номер стовпця.