Лабораторна робота №3

Умовні вирази, робота з математичними функціями

Мета роботи

- Ознайомлення з умовними та логічними виразами
- Вивчення принципів роботи умовних інструкцій

Короткі теоретичні відомості

Під час читання наступного теоретичного матеріалу, наполегливо радимо Вам повторювати усі наведені по ходу приклади та вправи.

Перевірте, що у Вас встановлений `Python` версії 2.7.5 і запускається його командний рядок (Python (command line)) – інтерпретатор:

```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

При запуску інтерпретатора ми бачимо інформацію про його версію, додаткову інформацію і запрошення `>>>` вводити оператори `Python`. У випадку використання `Interactive DeveLopment Environment (IDLE)` нам доступні додаткові зручності, зокрема у відображенні тексту програми на екрані.

1. Засоби керування логікою

Програми, які згадувалися в лабораторних роботах №1, 2 є простими. У більшості випадків будь-яка програма, яка робить щось корисне буде набагато складніша. Основною особливістю програмування є спроможність програми приймати рішення від імені людини, виконуючи інструкції, коли справджуються певні умови або послідовно обробляти текстові дані до тих пір поки не задоволена певна умова.

Наприклад є сенс шукати квадратний корінь з числа, тільки тоді коли воно більше 0. А як це сказати програмі? Або якщо ми хочемо піднести число до деякого ступеню. Якщо значення ступеню невелике, то ми просто можемо перемножити число само на себе. А що робити, якщо необхідно піднести число до 1000-ного ступеню? Звісно, можна перемножити послідовно число 1000 разів, але як при цьому не допустити помилку?

1.1 Умовні вирази

Python підтримує широкий набір операторів для встановлення взаємозв'язків між змінними (значеннями). Повний набір цих операторів наведений у таблиці 1.

Таблиця 1.

Оператор	**Значення**
<	Менше
<=	Менше або дорівнює
==	Дорівнює
!=	Не дорівнює
>	Більше
>=	Більше або дорівнює

Можна використовувати ці оператори безпосередньо:

```
>>> 3<5
True
>>> 5<3
False
>>> x = 3
>>> x < 5
True
>>> y = 5
>>> x < y
True
>>> x == y
False
>>> x = 5
>>> x == y
True
>>> x != y
False
>>> z = 3
```

```
>>> x != z
True
>>> x >= z
True
```

В результаті ми отримали вирази із булевими значеннями True, False.

1.2 Логічні вирази

Умовні (або логічні) вирази типу x >= 1023 є простими. Однак, на практиці не рідко використовуються більш складні. Може знадобитися отримати відповіді "Так" або "Ні" залежно від результату виконання двох простих виразів. Наприклад, "на вулиці йде сніг або дощ", "змінна new більше 12 і менше 20" і т.п.

У таких випадках використовуються спеціальні оператори, що об'єднують два і більше простих логічних вирази. Широко використовуються два способи об'єднання: через, так звані, логічне І (and) і АБО (or).

Щоб отримати істину (True) при використанні оператора and, необхідно, щоб результати обох простих виразів, які пов'язує даний оператор, були істинними. Якщо хоча б в одному випадку результатом буде False (неправда), то і весь складний вираз буде хибним.

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

Щоб отримати істину (True) при використанні оператора or, необхідно, щоб результати хоча б одного простого виразу, що входить до складу складного, був істинним. У разі оператора or складний вираз стає хибним лише тоді, коли хибні всі його складові.

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

Тепер можна визначати результати складних виразів:

```
x = 8
y = 13
x == 8 and y < 15 #  x равен 8 и y меньше 15
x > 8 and y < 15 #  x больше 8 и y меньше 15
x != 0 or y >15 #  x не равен 0 или y больше 15
x < 0 or y >15 #  x меньше 0 или y больше 15
```

Вправа: Визначте усно результати виконання операцій, наведених у прикладі вище. Перевірте правильність ваших припущень, виконавши дані вирази за допомогою інтерпретатора мови Python.

2. Умовна інструкція

Умовна інструкція (інструкція розгалуження) — це розвилка на шляху виконання програми. Програма може піти по одному шляху, а може і по іншому. Шлях вибирається в залежності від значення змінних в момент перевірки умови.

Наприклад, наступна програма друкує на екран найбільше з двох введених чисел:

```
>>> x=2
>>> y=5
>>> if x>y:                  # Галуження залежно від умови x>y
      print x                # Надрукуємо x якщо умова істинне
else:
      print y                # Надрукуємо y якщо умова помилкова


5
>>>
```

Ця програма перевіряє, яке з двох введених чисел більше (x>y). Якщо ця умова істинна, то друкується значення x, інакше значення y. Таким чином, дана програма надрукує найбільше з двох чисел.

Щоб виконати цю програму в інтерпретаторі, необхідно звернути увагу, на те, що після оператора *if x>y:* необхідно вказати де починається множина команд, які мають виконатись у разі x>y, і де вони закінчуються. Ця множина команд у мові Python вказується за допомогою відступів (кнопка Tab на клавіатурі). Це саме стосується і тексту програми, який зберігається у вигляді скрипта у файлі. Наприклад, у попередню програму додамо ще збереження максимально значення у змінну z:

```
>>> x = 2
>>> y = 5
>>> if x>y:
```

```
...     z = x
...     print x
... else:
...     z = y
...     print y
...
5
>>> print z
5
```

Особливістю мови Python є так зване "форматування відступами", яке використовується для групування команд (т.зв. "програмні дужки"). У мові Pascal аналогічну функцію виконують оператори begin ... end.

У загальному випадку синтаксис інструкції галуження такий:

```
if умова:
   інструкція11
   ...
   інструкція1n
else:
   інструкція21
   ...
   инструкция2n
```

Працює інструкція галуження так. Спочатку перевіряється умова. Якщо вона істинне (результат логічного виразу дорівнює True), то виконується та послідовність інструкцій, яка знаходиться між ключовими словами if і else (називатимемо цю послідовність інструкцій if-блоком). Якщо ж умова помилкова (результат логічного виразу дорівнює False), то замість цього блоку інструкцій виконується та послідовність, яка слідує після ключового слова else (else-блок). Після завершення виконання if-блоку або else-блоку, тобто незалежно від умови, починають виконуватися інструкції, наступні після else-блоку (у розглянутому прикладі це остання інструкція print).

2.1 Структура умовної інструкції

Як Python визначає, де закінчується `else-блок` і починаються ті інструкції, які будуть виконані незалежно від умови, що перевіряється, тобто в якому місці шляхи виконання зливаються?

Це визначається за величиною відступу: перед всіма інструкціями в if-блоці і else-блоці слід поставити один або кілька пропусків, виділяючи за допомогою зсуву вправо блок від решти програми, причому число пропусків перед всіма інструкціями в блоці повинне бути однаковим. Перша ж інструкція після else-блоку повинна починатися з тієї ж позиції, що і ключові слова if і else.

І в жодному випадку не слід забувати про двокрапки після інструкцій if і else.

2.2 Умови

У простому випадку умови мають наступний вигляд:

```
вираз1 оператор вираз2
```

де `вираз1` і `вираз2` – деякі арифметичні вирази (змінні, арифметичні оператори, виклики функцій тощо), а `оператор` може бути наступним оператором відношення:

```
<     менше
>     більше
<=    менше або рівно
>=    більше або рівно
==    рівно
!=    нерівно
```

Наприклад, умова `x <= 2**(0.5)` означає "значення змінної `x` не менше кореня з 2", а умова `2*x !=` у означає "подвоєне значення змінної `x` не рівно значенню змінної `y`".

Увага: оператор `==` (два знаки рівно) — це перевірка на рівність двох виразів, а оператор `=` (один знак рівно) — це привласнення змінній нового значення і використовувати його в умові інструкції `if` не можна.

Наприклад наступний скрипт виконує перевірку перед тим, як виконати математичний вираз:

```
x = int(raw_input("Please enter x: "))
if x == 2:
     print "Error: Division by zero"
else:
     y = (x + 2)/(x - 2)
     print y
```

Або приклад, де необхідно виконати одночасно декілька перевірок:

```
x = int(raw_input("Please enter x: "))
y = int(raw_input("Please enter y: "))
if x == 2 or y == -3:
     print "Error: Division by zero"
else:
     z = (x + 2)/((x - 2)*(y + 3))
     print z
```

2.3 Неповна інструкція галуження

У інструкції галуження може бути відсутнім ключове слово else з подальшим else-блоком. В наступній програмі ми створили змінну word, яка містить значення ‘cat’ типу стрічка. If- оператор перевіряє умову чи довжина слова <5, чи ні. Якщо умова виразу справджується, то виконується тіло if оператора і виконується оператор print .

```
>>> word='cat'
>>> if len(word)<5:
      print "word lenght is less than 5"


word lenght is less than 5
>>>
```

Якщо змінити умови виразу (довжина слова більша або рівна 5), вираз не справджується і оператор print не виконується.

```
>>> if len(word)>=5:
      print "word lenght is greater than or equal to 5"


>>>
```

2.4. Інструкція множинного галуження

Іноді потрібно вибирати не з двох альтернативних шляхів виконання, а з кількох, наприклад, в залежності від того, чи є певна величина додатною, від’ємною чи рівною нулю слід виконати одну з трьох дій. В цьому випадку можна використовувати *інструкцію множинного галуження*. Приклад використання інструкції множинного галуження:

```
>>> if len(word)>=5:
      print "word lenght is greater than or equal to 5"
else:
      print "word lenght is less than 5"


word lenght is less than 5
>>>
```

У множинному галуженні повинна бути одна інструкція `if` з подальшим блоком, одна або кілька інструкцій `elif` з подальшими блоками і, можливо, інструкція `else` з подальшим блоком. Всі умови перевіряються одна за іншою і буде виконаний блок інструкцій, який слідує за умовою, значення якої буде істинне.

Наступний приклад містить складнішу комбінацію з операторів *if, else, elif.*

```
x = int(raw_input("Please enter x: "))
y = int(raw_input("Please enter y: "))
if x == 2:
     print "Error: x couldn't be equal 2 - Division by zero"
elif y == -3:
     print "Error: y couldn't be equal -3 - Division by zero"
else:
     z = (x + 2)/((x - 2)*(y + 3))
     print z
```

Якщо істинними виявляться кілька умов, то виконається тільки той блок інструкцій, який слідує після першого з цих умов (а решта умов навіть не перевірятиметься).

```
>>> x = 2
>>> y = -3
>>> if x == 2:
     print 1
elif y == -3:
     print 2

1
>>>
```

У випадку, коли *if.* вираз твердження задовольняється, *elif* вираз не виконується і програма ніколи не виведе на екран 2. *elif* вираз надає більше інформації ніж *if* вираз. Якщо *elif* вираз справджується то це означає, що не тільки одна умова справджується але і означає, що умова *if* виразу не справдилася.

Якщо ж всі умови будуть помилкові, то виконається else-блок, якщо він є.

2.5. Вкладені умовні інструкції

Усередині блоку умовної інструкції можуть знаходиться будь-які інші інструкції, у тому числі і умовна інструкція. Такі інструкції називаються *вкладеними*. Синтаксис вкладеної умовної інструкції такий:

```
if умова1:
   ...
   if умова2:
      ...
   else:
      ...
```

```
        ...
    else:
        ...
```

Замість трикрапок можна писати довільні інструкції. Зверніть увагу на розміри відступів перед інструкціями. Блок вкладеної умовної інструкції відділяється великим відступом. Рівень вкладеності умовних інструкцій може бути довільним, тобто всередині однієї умовної інструкції може бути друга, а всередині неї — ще одна і т.д.

3. Поняття модуля

Працюючи в інтерактивному режимі інтерпретатора Python неможливо писати великі програми, оскільки після виходу з інтерпретатора визначення функцій “зникають” безслідно. А хотілося б мати можливість написати кілька функцій в одному файлі і потім мати можливість звертатися до них з інших програм (принцип повторного використання коду). Така можливість в Python існує. Це технологія модулів, тобто текстових файлів, що містять в собі який-небудь набір функцій(бажано об'єднаних за якою-небудь ознакою). Більшість функцій стандартної бібліотеки Python реалізована саме в модулях.

Модулі можуть містити в собі окрім визначень функцій і деякий виконуваний код. Цей код не входить в жодну функцію і виконується інтерпретатором, коли даний модуль підключається ключовим словом import.

Python поставляється з безліччю програмних модулів, покликаних полегшувати роботу програміста. Ці модулі є стандартними для всіх реалізацій Python, тому використання стандартних функцій допомагає зробити код сумісним з іншими платформами.

В Python є також значна кількість вбудованих функцій, що забезпечують найнеобхіднішу роботу. У бібліотеці Python є також модулі, які специфічні для конкретної операційної системи і забезпечують високоефективний доступ до її ресурсів. Використання таких бібліотек повністю виключено, якщо ви хочете писати переносимий код.

Модулі підключаються до програми (або іншого модуля) з допомогою оператора `import`, після якого імена з простору імен модуля стають доступними.

Які саме імена стають доступні, визначає оператор `import`: варіант `import module` робить доступним рівне одне ім'я - ім'я модуля `module`, та зате через це ім'я можна використовувати всі глобальні імена модуля у вигляді `module.name`.

У варіанті `from module import name` з модуля імпортується вказане ім'я або список імен. У варіанті `from module import *` з модуля імпортуються всі імена. Хоча автор модуля може обмежити цей список, а у відсутності такого обмеження не імпортуються імена, що починаються з підкреслення, - вважається, що це внутрішні імена модуля, що не входять до його публічного інтерфейсу.

Наприклад, щоб підключити модуль у якому містяться математичні функції необхідно набрати наступні команди:

```
>>> import math
>>> math.sqrt(9)
3.0
>>> math.cos(3)
-0.9899924966004454
```

Або, щоб не доводилось кожен раз набирати слово math

```
>>> from math import *
>>> sqrt(9)
3.0
>>> cos(3)
-0.9899924966004454
```

Завдання та порядок виконання роботи

1. Ознайомитися з наведеними вище теоретичними відомостями.
2. Виконати приклади, які приводяться в теоретичних відомостях.
3. Виконати наступні завдання:

   3.1. Написати скрипт, який:

   - на початку, у вигляді коментаря, буде містити назву курсу та номер лабораторної роботи, а також ваше ім'я та прізвище, та номер Вашої заліковки
   - у першому рядку буде виводити назву курсу та номер лабораторної роботи
   - у другому – буде виводити ваше ім'я та прізвище, а також номер Вашої заліковки.
   - далі, просити ввести з клавіатури (`raw_input`) необхідні значення змінних для обчислення виразу, відповідно до варіанту, та виводить результат обчислення при даних змінних
   - Якщо значення змінних виходять за область визначення функції, то замість результату виразу має вивестись повідомлення про це, а сам вираз не має обчислюватись
   - Довідник по математичним функція у модулі math можна подивитись тут http://bit.ly/18jMBaD

1) $y = \frac{\sin(2a)}{a-3} + \frac{arctg\, b}{c}$;

2) $y = \log_2(x-4) + e^{2a-x}$;

13) $y = \ln(x^3 - 8) + \frac{1}{\sin a}$;

14) $z = \frac{1}{x^3 - y^3} - \sqrt{2x}$;

3) $y = \frac{2x^2 + x - 5}{x + 2} + ctg\frac{x}{2z}$;

4) $y = \frac{x^3 + 2ax + 3}{(x-1)^2} + \frac{\cos x^2}{a+2}$;

5) $y = \sqrt{25 - x^2} + \frac{2a}{x-3}$;

6) $y = \frac{a-2}{2a+b} + \frac{\sin 3a}{\cos b}$;

7) $z = \frac{5x}{x^3 + y^3} - ctg\frac{3x}{y}$;

8) $z = \sqrt{x^3 - y^3} + \log_{10}(x+5)$;

9) $y = \frac{2x - c}{\sqrt{x-b}} + |x - c|$;

10) $y = \frac{x + e^{z-1}}{1 - x^2|x - z|}$;

11) $y = \sqrt{\ln(x^2 - 4)}$;

12) $y = \frac{\sqrt{|x^2 - 4|}}{2z}$;

15) $y = \frac{\sqrt{x-b}}{2b} - \frac{tg\,x}{b^2}$;

16) $y = \frac{x + 2x + 3}{z - 2} + arctg\,z$;

17) $y = \sqrt{\frac{(a-b)^5}{(b-a)^3}}$;

18) $y = \frac{|x^3 - z^3|}{\sqrt{x^2 - 9}}$;

19) $y = \frac{\sqrt{|x-1|} - \sqrt[3]{|z|}}{1 - \frac{x^2}{2} - \frac{z}{x}}$;

20) $y = z + \frac{x}{z^2 - \left|\frac{x^2}{z - x^3/3}\right|}$;

21) $y = \log_4\left|(z - \sqrt{|x|})\right|$;

22) $y = x - \frac{z}{z - x^2/4} - \frac{x^2}{5!}$;

23) $y = x\,arctg\,z + e^{-\frac{x+3}{z-2}}$;

4. Зберегти скрипт у файл з наступною назвою <Surname>_Task3.py (наприклад Rodionov_Task3.py)

5. Запустити даний скрипт за допомогою інтерпретатора та переконатись у його працездатності та правильності результатів.

6. Спробувати здати та захистити лабораторну роботу у викладача практичних занять

Література

1. A Byte of Python (Russian) (http://wombat.org.ua/AByteOfPython/#id14)

2. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с..

3. Computational Physics with Python by M. Newman (http://www-personal.umich.edu/~mejn/computational-physics/)

Інтернет посилання

- Библиотека math: http://bit.ly/18jMBaD
- http://ru.wikipedia.org/wiki/Python
- MIT: Introduction to Computer Science and Programming
- http://python.org