Лабораторна робота №2

Робота зі змінними, типи змінних

Мета роботи

- Ознайомлення з основними типами даних в Python.
- Вивчення основ програмування на мові Python.

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. Змінні, операції і вирази

1.1. Значення і типи

Всі програми працюють із значеннями. Значенням може бути число або рядок.

Наприклад, в першій програмі ми вже друкували на екрані стрічкове значення `"Hello world!"`. Аналогічним чином можна вивести і число:

```
>>> print 12
12
```

`"Hello world!"` і `12` належать до різних типів: `str` (від англ. string – рядок) і `int` (від англ. integer – ціле число). В деяких версіях `Python` ці типи носять нескорочені назви: string і integer. Інтерпретатор відрізняє рядок від числа по лапках, в які вона поміщена.

Якщо ви не впевнені в тому, до якого типу належить значення, це можна перевірити так:

```
>>> type("Hello world!")
<type 'str'>
>>> type(12)
<type 'int'>
>>> type(2.4)
<type 'float'>
>>> type(12/2)
<type 'int'>
>>> type(5/2)
<type 'int'>
>>> type(5.0/2)
<type 'float'>
>>> type(5/2.0)
<type 'float'>
```

Стрічковий тип називається в `Python` str, цілочисельний носить назву int, а дріб – float (від англ. floating-point number – число з плаваючою точкою).

Зверніть увагу на три останні приклади. Якщо 5/2 то результуючий тип буде int, але якщо написати 5.0/2 або 5/2.0, то результуючий тип буде float. Від результуючого типу буде залежати і результат операції ділення:

```
>>> type(5/2)
<type 'int'>
>>> 5/2
2
>>> type(5.0/2)
<type 'float'>
>>> 5.0/2
2.5
```

- ***Вправа.*** Проведіть самостійно наступний експеримент: перевірте типи значень "12" і "2.4"? Якого вони типу і чому?

- ***Вправа.*** Що відбудеться, якщо стрічку "8.53" спробувати перетворити в ціле число за допомогою функції `int()`? Як вирішити цю проблему?

1.2. Перетворення типів

Функція `int()` перетворює значення в цілочисельний тип. Якщо перетворення провести неможливо, то буде виведено повідомлення про помилку:

```
>>> int("32")
32
>>> int("Hello")
Traceback (most recent call last):
File "", line 1, in ?
ValueError: invalid literal for int(): Hello
>>> int("8.53")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '8.53'
```

Функція `int()` може приводити до цілого типу і дробові числа, але не забувайте, що при перетворенні вона просто відкидає дробову частину:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

Функція `float()` перетворює цілі числа і стрічки в дробовий тип:

```
>>> float(32)
32.0
>>> float("3.14159")
3.14159
```

І, нарешті, функція `str()` відповідає за перетворення до стрічкового типу. Саме її заздалегідь запускає команда `print`:

```
>>> str(32)
'32'
>>> str(3.14149)
'3.14149'
```

`Python` розрізняє ціле число 1 від дробового 1.0: це одне і те ж число, але воно належить до різних типів. Від типу значення залежить спосіб його зберігання в пам'яті.

1.3. Змінні

Як будь-яка інша мова програмування Python підтримує концепцію змінних, але з невеликою відмінністю. Якщо в мовах C++ або Pascal змінна – це ім'я елементу пам'яті, в якій зберігається значення, то в Python змінна – це посилання на комірку пам'яті. Відмінність, на перший погляд, неістотна, але насправді це трохи іншої підхід до організації зберігання об'єктів в пам'яті.

Для того, щоб «запам'ятати» значення достатньо привласнити його змінній. Це робиться за допомогою спеціального оператора привласнення, який позначається знаком рівності (=).

```
>>> message = "Hello world!"
>>> n = 12
>>> pi = 3.14159
```

У даному прикладі змінній message привласнюється (або зіставляється) значення "Hello world!", змінній n привласнюється значення 12, а змінній pi – 3.14159.

Команда print працює і зі змінними:

```
>>> print message
Hello world!
>>> print n
12
>>> print pi
3.14159
```

Як бачите, команда print виводить не імена змінних, а їх значення. Змінні так само як і значення, мають тип. Давайте це перевіримо за допомогою функції type():

```
>>> type(message)
<type 'str'>
>>> type(n)
<type 'int'>
```

Тип змінної співпадає з типом привласненого нею значення. Розглянемо ще один приклад:

```
>>> message = "Hello world!"
>>> n = 12
>>> type(message)
<type 'str'>
>>> message = n
>>> print message
12
>>> type(message)
<type 'int'>
```

Цей приклад є цікавим з двох причин. По-перше, в ньому використана можливість привласнювати значення однієї змінної іншій. Конструкція `message=n` працює аналогічно привласненню змінній значення: змінній `message` привласнюється значення змінної n. При цьому значення 12 зберігається в пам'яті тільки один раз – `Python` досить економно витрачає пам'ять.

По-друге, як видно з прикладу, змінна `message` після привласнення їй значення `n` поміняла свій тип. Далеко не кожна мова програмування «вміє» це робити так просто (у мовах C/C++, Java таке не спрацює).

1.4. Імена змінних і ключові слова

Для перетворення абстрактної, не проявленої в матеріальному світі ідеї в код програми необхідно виділити сутність і дії, придумати їм назви, щоб ними управляти, прослідкувати зв'язки між ними та їх властивостями. Ви, напевно, здогадалися, що іменами сутностей, з якими працює програміст, служать змінні. Тому варто вибирати осмислені назви змінних.

Імена змінних можуть бути довільної довжини, але прагніть вибирати не дуже короткі і не дуже довгі імена – від цього залежить читабельність програми.

При визначенні імен змінних в `Python` можна використовувати будь-які латинські букви, цифри і знак _ (знак підкреслення). Знак підкреслення може використовуватися для розділення слів складових ім'я змінної: наприклад, `user_name` або `full_price`. Але назви змінних не можуть починатися з цифри.

```
>>> 1message = "Hello world!"
File "<stdin>", line 1
1message = "Hello world!"
^
SyntaxError: invalid syntax

>>> price_in_$ = 300
File "<stdin>", line 1
price_in_$ = 300
^
SyntaxError: invalid syntax

>>> class = "Computer Science 101"
File "<stdin>", line 1
class = "Computer Science 101"
^
SyntaxError: invalid syntax
```

Розберемо ці три приклади. Перший вираз інтерпретатору не сподобався, і він відзначив знаком ^, де саме у нього виникли претензії: він вказав на найменування змінної `1message`. Дійсно, ім'я `1message` є некоректним, адже воно починається з цифри. Аналогічна ситуація з іменем `price_in_$`: воно містить неприпустимий символ $. Але що інтерпретатору не подобається в третьому виразі? Давайте спробуємо змінити ім'я змінної `class` на що-небудь схоже, наприклад, `class_`:

```
>>> class_ = "Computer Science 101"
>>> print class_
Computer Science 101
```

Тепер все гаразд. У чому ж справа? Чому ім'я `class` викликало помилку, а ім'я `class_` – ні? Які є припущення? Поставимо ще один експеримент:

```
>>> print = "Some message"
File "<stdin>", line 1
print = "Some message"
^
SyntaxError: invalid syntax
```

Знайома ситуація, чи не так? Проаналізуємо те, що ми отримали. Як ім'я змінної ми намагалися використовувати команду `print` і отримали аналогічну помилку, тому слово `class`, швидше за все, теж є командою або якимсь службовим словом.

Дійсно, слова `class` і `print` є так званими ключовими словами.

Всього в Python зарезервовано 29 ключових слів:

`and`	`del`	`for`	`is`	`raise`
`assert`	`elif`	`from`	`lambda`	`return`
`break`	`else`	`global`	`not`	`try`
`class`	`except`	`if`	`or`	`while`
`continue`	`exec`	`import`	`pass`	`yield`
`def`	`finally`	`in`	`print`	

Корисно мати цей список під рукою, щоб заглянути в нього, коли раптом інтерпретатору не сподобається одне з імен змінних.

Слід пам’ятати, що інтерпретатор розрізняє великі і маленькі букви, тобто `message` і `Message` будуть різними змінними.

- ***Вправа.*** Напишіть програму, яка підтверджує, що інтерпретатор `Python` розрізняє рядкові і заголовні букви в іменах змінних.

1.5. Вирази

Вираз – це послідовність синтаксичних одиниць, що описує елементарну дію мовою програмування.

Наприклад, `print "Hello world!"` і `message = n` є виразами.

Коли ви набираєте вираз в командному рядку, інтерпретатор виконує його і виводить результат, якщо він існує. Результатом виразу `print "Hello world!"` є рядок: `Hello world!`. Вираз привласнення нічого не виводить.

1.6. Виконання виразів

По суті, **вираз** – це послідовність значень, змінних і операторів.

Якщо ви напишете вираз, то інтерпретатор, після виконання, виведе його на екран:

```
>>> 1 + 1
2
```

Значення само по собі розглядається як вираз, так само як і змінна:

```
>>> 17
17
>>> x = 2
>>> x
2
```

Але виконання і виведення результату виконання виразу не зовсім те ж саме:

```
>>> message = "Hello world!"
>>> message
"Hello world!"
>>> print message
Hello world!
```

Коли `Python` виводить значення виразу в командному режимі, він використовує той же формат, що використовується при введенні цього значення. Наприклад, при виведенні рядків він бере їх в лапки. Команда `print` також виводить значення виразу, але у випадку з рядками, вона виводить вміст рядка без лапок.

У командному режимі інтерпретатор `Python` зберігає результат останнього виразу в спеціальній змінній _ (знак підкреслення). Ви можете переглянути результат виконання останнього виразу і використовувати його в своїх виразах:

```
>>> 1.25 + 1
```

```
2.25
>>> print _
2.25
>>> 5 + _
7.25
```

1.7. Оператори і операнди

Операторами називають спеціальні символи (або послідовності символів) що позначають певні операції. Наприклад, знаком «+» позначають операцію додавання, а знаком «*» – множення. Значення, над якими виконується операція, називають **операндами**.

Всі нижченаведені вирази, з погляду `Python` є коректними:

```
20+32     hour-1    hour*60+minute minute/60 5**2 (5+9)*(15-7)
```

Значення більшості з них неважко зрозуміти. Значення символів +, - * і / у `Python` такі ж, як в математиці. Дужки використовуються для групування операцій, а двома зірочками (**) позначається операція піднесення до ступеня.

Якщо операндом є змінна, то перед обчисленням виразу проводиться підстановка на її місце значення, на яке вказує ця змінна.

Додавання, віднімання, множення і піднесення до ступеня працюють в звичний для нас способом, але дія операції ділення дещо відрізняється. Це ілюструє наступний приклад:

```
>>> minute = 59
>>> minute/60
0
```

Значення змінної `minute` рівне 59; результат ділення 59 на 60 повинен бути 0.98333, а не 0. Причиною цієї невідповідності є те, що `Python` виконує цілочисельне ділення.

Коли обидва операнди – цілі числа, і `Python` вважає, що результат теж повинен бути цілим. Тому цілочисельне ділення завжди відкидає дробову частину.

Як отримати дробовий результат? Достатньо примусово перетворити один з операндів в дробове число:

```
>>> minute = 59
>>> float(minute)/ 60
0.983333333333
```

Інший варіант:

```
>>> minute = 59
>>> minute / 60.0
```

```
0.983333333333
```

Якщо один з операндів належить до типу `float`, то другий автоматично перетворюється до цього типу, як до складнішого.

1.8. Порядок операцій

Більшість мов програмування дозволяють групувати в одному виразі кілька операцій. Це зручно, наприклад, якщо потрібно порахувати процентне співвідношення двох величин:

```
>>> print 100 * 20 / 80, "%"
25 %
```

В даному прикладі обчислюється процентне співвідношення двох чисел: 20 і 80. Після результату виразу виводиться символ % – інтерпретатор обчислює арифметичне вираз і виводить результат, а потім дописує рядок, що стоїть після коми.

Коли у виразі є більш ніж один оператор, послідовність виконання операцій залежить від порядку їх проходження у виразі, а також від їх пріоритету. Пріоритети операторів в `Python` повністю співпадають з пріоритетами математичних операцій.

Найвищий пріоритет у дужок, які дозволяють змінювати послідовність виконання операцій. Таким чином, операції в дужках виконуються в першу чергу.

Наприклад, `2*(3-1)` рівне `4` `(1+1)**(5-2)` `- 8`. Дужки зручно використовувати і для того, щоб вирази було легко читати, навіть якщо їх наявність у виразі ніяк не відбивається на результаті: `(100*20)/80`.

Наступний пріоритет у операції піднесення до ступеня, тому `2**1+1` рівне 3, а не 4, і вираз `3*1**3` дасть результат 3, а не 27.

Множення і ділення мають однаковий пріоритет, вищий, ніж в операцій додавання і віднімання. `2*3-1` дорівнює 5, а не 4; `2/3-1` дорівнює -1, а не 1 (результат цілочисельного ділення 2/3=0).

Оператори з однаковим пріоритетом виконуються зліва направо. Отже в виразі `100*20/80` множення виконується першим (виразу набуває вигляд 2000/80); потім виконується ділення, значення, що видає в результаті, 25. Якби операції виконувалися справа наліво, то результат вийшов би іншим.

- ***Вправа***. Змініть вираз `100*20/80` так, щоб послідовність виконання операцій була зворотною. Який результат ви отримали після його виконання і чому?

2. Ввід з клавіатури

Функція **raw_input ('вітання')** зупиняє виконання програми, друкує вітання і чекає введення з клавіатури, що завершується натисканням **<Enter>**. При цьому функція за замовченням повертає набране користувачем у вигляді рядка (str):

```
>>> a = raw_input('Input please: ')
Input please: please
>>> a
'please'
>>> a = raw_input('Input please: ')
Input please: 10
>>> print a
10
>>> type(a)
<type 'str'>
```

Таким чином, ми не можемо виконувати арифметичні операції зі змінною a, навіть якщо ввели чисельне значення:

```
>>> a = raw_input('Input please: ')
Input please: 10
>>> a + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> print a
10
>>> type(a)
<type 'str'>
```

Що робити? Адже ми не збираємось працювати виключно з рядками. І тут як раз у нагоді стає перетворення типів (з розділу 1.2). Ми можемо записати наступний вираз **a = int(a)** (щоб не створювати додаткову змінну):

```
>>> a = raw_input('Input please: ')
Input please: 10
>>> type(a)
<type 'str'>
>>> a = int(a)
>>> type(a)
<type 'int'>
>>> a + 2
```

```
12
```

Або, щоб було зручніше, одразу спробувати перетворити результат функції до необхідного нам типу:

```
>>> a = int(raw_input('Input integer number: '))
Input integer number: 888
>>> a
888
>>> f = float(raw_input('Input real number: '))
Input real number: 3.14
>>> f
3.14
```

Далі змінні a та f можна сміливо використовувати у арифметичних виразах.

```
>>> a = int(raw_input('Input integer number: '))
Input integer number: 3
>>> f = float(raw_input('Input real number: '))
Input real number: 3.5
>>> c = (a+2)/(f-3)
>>> print c
10.0
```

3. Що таке відлагодження?

Програмування – досить складний процес, і цілком природно, коли програміст припускається помилки. Так повелося, що програмні помилки називають «багами» (від англ. bug – жучок). В сленгу програмістів воно використовується достатньо часто разом із словом «глюк». Процес пошуку і усунення помилок в англомовній літературі прийнято позначати терміном debugging ми ж називатимемо його відлагодженням.

Існує три типи помилок, які можуть виникнути в програмах:

- синтаксичні помилки (syntax errors),
- помилки виконання (runtime errors),
- семантичні помилки (semantic errors).

Щоб знаходити і виправляти їх швидше, має сенс навчитися їх розрізняти.

3.1. Синтаксичні помилки (syntax errors)

Будь-який інтерпретатор зможе виконати програму лише в тому випадку, якщо програма є синтаксично правильною. Відповідно, компілятор теж не зможе перетворити програму в машинні інструкції, якщо програма містить синтаксичні помилки. Коли транслятор знаходить помилку (тобто доходить до інструкції, яку не може зрозуміти), він перериває свою роботу і виводить повідомлення про помилку.

Для більшості читачів синтаксичні помилки не представляють особливої проблеми. Наприклад, часто зустрічаються вірши без розділових знаків, але ми без зусиль можемо їх прочитати, хоча це часто породжує неоднозначність їх інтерпретації. Але транслятори (і інтерпретатор Python не виключення) є дуже прискіпливими до синтаксичних помилкам.

Навіть якщо у програмі Python знайде хоча б незначну друкарську помилку, він тут же виведе повідомлення про те, де він на неї наткнувся, і завершить роботу. Таку програму він не зможе виконати і тому відкине. У перші тижні вашої практики розробки програм ви, швидше за все, проведете досить багато часу, розшукуючи синтаксичні помилки. У міру накопичення досвіду ви допускатиме їх все рідше, а знаходити – все швидше.

3.2. Помилки виконання (runtime errors)

Другий тип помилок зазвичай виникає під час виконання програми (їх прийнято називати винятковими ситуаціями або, коротко – виключеннями, по-англійськи exceptions). Такі помилки мають іншу причину. Якщо в програмі виникає виключення, то це означає, що по ходу виконання відбулося щось непередбачене: наприклад програмі було передано некоректне значення, або програма спробувала розділити якесь значення на нуль, що неприпустимо з погляду дискретної математики. Якщо операційна система надсилає запит на негайне завершення програми, то також виникає виключення. Але в простих програмах це достатньо рідкісне явище, тому можливо, з ними ви зіткнетеся не відразу.

3.3. Семантичні помилки (semantic errors)

Третій тип помилок – семантичні помилки. Першою ознакою наявності у вашій програмі семантичної помилки є те, що вона виконується успішно, тобто без виняткових ситуацій, але робить не те, що ви від неї чекаєте.

У таких випадках проблема полягає в тому, що семантика написаної програми відрізняється від того, що ви мали на увазі. Пошук таких помилок – завдання нетривіальне, оскільки доводиться переглядати результати роботу програми і розбиратися, що програма робить насправді.

3.4. Процес відлагодження

Уміння відлагоджувати програми є дуже важливим навиком для програміста. Процес відлагодження вимагає великих інтелектуальних зусиль і концентрації уваги, але є одним з найцікавіших занять.

Відлагодження дуже нагадує роботу природодослідника. Вивчаючи результати свого попереднього експерименту, ви робите певні висновки, потім відповідно до них змінюєте програму, запускаєте її, і знову приступаєте до аналізу отриманих результатів.

Якщо отриманий результат не співпадає з очікуваним, то вам доведеться знову розбиратися в причинах, які спричинили ці невідповідності. Якщо ж ваша гіпотеза виявиться правильною, то ви зможете передбачити результат модифікацій програми і на крок наблизитися до завершення роботи над нею або, мабуть, це примусить вас ще більше увірувати в свою помилку.

Тому, для перевірки працездатності програми не достатньо перевіряти її один раз – потрібно придумати всі можливі набори вхідних даних, які можуть якось вплинути на стійкість вашої системи. Такі набори вхідних даних називають граничними значеннями.

Іноді процес написання і відлагодження програм розділяють не тільки в часі, але і між учасниками команди розробників. Але останнім часом великої популярності набувають так звані гнучкі методології розробки. В них кодування не відмежується від відлагодження: програмісти, що пишуть код, також відповідають і за підготовку тестів і виявлення як можна більшої кількості помилок вже в процесі кодування. Це дозволяє їм повною мірою насолодитися своєю роботою.

Отже, програмування – це процес поступового доопрацювання і відлагодження доти, поки програма не робитиме те, що ми хочемо. Починати варто з простої програми яка робить щось просте, а потім можна приступати до нарощування її функціональності, роблячи невеликі модифікації і відлагоджувати додані фрагменти коду.

Таким чином, на кожному кроці у вас буде працююча програма, що, в якійсь мірі дозволить вам судити том, яку частину роботи ви вже зробили.

4. Створення скриптів та коментарі

4.1. Створення скриптів

Не дивлячись на зручність використання інтерактивного режиму роботи, часто потрібно зберегти початковий програмний код для подальшого використання. В такому разі готуються файли з програмним кодом, які передаються інтерпретатору на виконання. По відношенню до мов програмування, що інтерпретуються, часто початковий код називають скриптом. Файли з кодом на Python зазвичай мають розширення *.py.

Підготувати скрипти можна в середовищі IDLE. Для цього, після запуску середовища в меню потрібно вибрати команду File → New Window (Crtl + N), відкриється нове вікно.

Потім бажано відразу зберегти файл з розширенням *.py. командою File→Save As. За замовченням, файл буде збережено в корні C:\Python27. Після того, як код буде написано, слід знов зберегти файл.

Увага: якщо набирати код, не зберігши файл на початку, то не буде здійснюватися підсвічування синтаксису.

Для запуску скрипту потрібно виконати команду меню Run → Run Module (F5). Після цього в першому вікні (де "працює" інтерпретатор) з'явиться результат виконання коду.

Насправді скрипти можна писати в будь-якому текстовому редакторі (бажано, щоб він підтримував підсвічування синтаксису мови Python). Також існують спеціальні програми для розробки, що надають додаткові можливості і зручності.

Запускати підготовлені файли можна не тільки в IDLE, але і в консолі за допомогою команди python адрес/имя_файла.

Крім того, існує можливість налаштувати виконання скриптів за допомогою подвійного кліка по файлу (у Windows дана можливість присутня спочатку).

4.2. Коментарі в програмах

У міру збільшення розмірів ваших програм рано чи пізно ви зіткнетеся з однією проблемою: їх стане складніше читати. У ідеалі програма повинна читатися так само легко, неначебто вона була написана на природній мові. Тому для підвищення зрозумілості коду, його корисно забезпечувати коментарями на природній мові.

Коли над програмою працює один програміст, то відсутність коментарів компенсується хорошим знанням коду, але при роботі в команді, за рідкісними виключеннями, коментарі просто необхідні. Крім того, через якийсь час важко розібратися в своїй програмі, якщо в ній не буде додаткових зауважень.

В Python коментарі позначаються символом **#** – рядки, що починаються з цього символу, просто ігноруються інтерпретатором і ніяк не впливають на її трансляцію і виконання. **Не** допускається використання перед символом **#** лапок:

```
>>> a = "Це рядок"    #Це коментар
>>> b = "#Це вже НЕ коментар"
```

Щоб створити коментар у скрипті, або зробити деякий рядок скрипта не виконуваним, достатньо на його початку поставити символ #.

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:

1) $\frac{11+2*x+4.1}{12.4-y}+z$; 2) $\frac{(z-23.1)*(2^x+2.2)}{y-21.1}-3.2$; 3) $\frac{x}{z-11.3}+\frac{y}{z+11.3}$;

4) $x+\frac{11-x/y}{87-z}$; 5) $\frac{x}{x+11.3}+\frac{y}{z-11.3}$; 6) $x-\frac{x+y/z}{78+y}$;

7) $\frac{15+x/y}{33+x}-z$; 8) $\frac{(12^x+17.1)*(y-254.2)}{17.4-z}-4.2$; 9) $\frac{x}{y-211.3}-\frac{z+11.3}{x}$;

10) $\frac{15-x/y}{33^y+19.3}+z$; 11) $\frac{x+z/8}{18.2-3.8^y+19.3}$; 12) $\frac{176-y/7}{18.2+x}-23.8^z$;

4. Зберегти скрипт у файл з наступною назвою <Surname>_Task2.py (наприклад Rodionov_Task2.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

- http://ru.wikipedia.org/wiki/Python
- MIT: Introduction to Computer Science and Programming
- http://python.org