Отладка программ

Аннотация

В уроке рассматривается работа с отладчиком на примере программы, которую нужно исправить. Затем даются группы задач, алгоритмы решения которых требуют рассмотрения многих случаев. В этом уроке нет тестов к задачам, точно так же, как на контрольных и самостоятельных работах. Вам придется самим выяснять, на каких входных данных не работает ваша программа. Внимательно читайте условие.

1.Типы ошибок

Вероятно, вы уже обратили внимание, что при написании программ у вас возникали хоть и разные ошибки, но у многих из них было что-то общее. На самом деле существуют три основных типа ошибок, которые нужно научиться различать, так вы сможете без труда их исправить.

Синтаксическая ошибка: происходит, когда встречается код, который не соответствует правилам языка Python. Например: отсутствие кавычек вокруг строковой переменной, невыставленное двоеточие или отсутствие отступов после оператора `if`. Интерпретатор останавливается и сообщает об ошибке без дальнейшего выполнения программы.

Ошибка исполнения: как следует из названия, случается во время исполнения программы. Например, когда переменная ожидает работу с целыми числами, а оказывается строкой. Из-за несоответствия типов интерпретатор останавливается на ошибке.

Логическая ошибка (смысловая): происходит, когда программа ведет себя не так, как было задумано. Например, порядок действий в вычисляемом выражении задан неправильно или неверно были расставлены отступы во вложенных операторах условия и т. д. Интерпретатор запускает программу и не сообщает об ошибке. Но все работает не так, как хотелось бы.

С исправлением первых двух типов ошибок справиться достаточно просто, поскольку интерпретатор сообщает, где ошибка произошла, а в среде разработки чаще всего указывается строка, на которой эта

ошибка возникла. А для логических ошибок требуется детальное изучение кода, чтобы найти такие ошибки, разработан специальный механизм отладки.

2.Отладка

Отладка

Процесс изучения и исправления ошибок в работе программы (их часто называют багами, от англ. bug — жук), называется **отладкой**, по-английски — debugging, одно из значений — «удаление насекомых с растений». По одной из версий, именно так приходилось чинить компьютеры на заре компьютерной эпохи.

Программист всегда тестирует программу, занимается трассировкой и, конечно, размышляет над кодом, но иногда этого недостаточно. Часто (особенно когда программа длинная и сложная) невозможно обойтись без дополнительной информации. Какое значение имеет такая-то переменная в такой-то момент работы программы? В таком-то месте выполняется `if` или `else`? Выполняется ли тело такого-то цикла и, если да, сколько раз?

Для ответа на такие вопросы существуют специальные инструменты отладки (**отладчики**). Они позволяют проходить программу пошагово и следить при этом за значениями переменных.

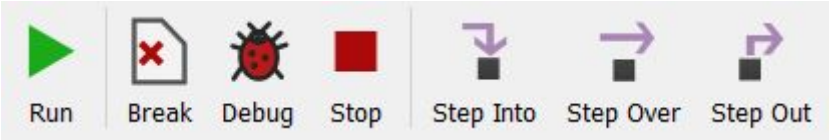
Средства отладки есть в стандартной библиотеке функций Python, но обычно удобнее пользоваться возможностями среды программирования. Рассмотрим возможности отладчика Wing IDE на примере программы из прошлого урока.

```
total = 0

print('Вводите цены; для остановки введите -1.')

price = float(input())

while price > 0:

    total = total + price  # можно заменить на аналогичную запись

    # total += price

    price = float(input())

print('Общая стоимость равна', total)
```

Возьмем вот эту программу. Запустим ее, убедимся, что она работает.

Попробуем пройти по ней в режиме отладки.

Средства отладки расположены в меню **Debug** и выведены иконками (после кнопки запуска) в панели меню. Еще их можно запускать горячими клавишами.



В меню **Debug** есть такие команды (обратите внимание и на горячие клавиши, указанные справа):

-Start / Continue и Stop Debugging. Кнопки Debug и Stop выполняют те же функции. Первая команда запускает выполнение программы до ближайшего breakpoint'а, вторая полностью прерывает процесс отладки.

-Step Into, Step Over, Step Out, Step Out to Here. Управление пошаговым прохождением программы. Мы пока будем использовать только Step Over. Остальные пункты связаны с заходом внутрь функций и выходом из них, но мы свои функции пока не пишем, а заходить внутрь стандартных нам не нужно. Пункт Step Over (как и одноименная кнопка) доступен, когда программа запущена, но остановлена по breakpoint'у. Нажатие этой кнопки выполняет команду, записанную на текущей строчке, переходит на следующую и останавливается, ожидая еще одного Step Over либо Continue.

-Из следующих пунктов нам пока понадобятся только Add Breakpoint и Remove All Breakpoints. Add Breakpoint, как и кнопка Break, добавляет точку остановки на ту строчку программы, где находится курсор.

Важно!

Если вы не видите подписей под кнопками, их можно включить: Edit → Preferences... → User Interface → Toolbar → Toolbar Style → Icon and Text Below.

-Запустим отладку, выбрав из меню команду Step Into (или одноименную кнопку или воспользуемся горячей клавишей F7). После запуска этой команды вместо окна Python Shell стало активно окно Debug I/O. В этом окне будет происходить ввод/вывод данных при

отладке. Первая строка программы выделилась розовым цветом, и у номера строки появилась розовая стрелка. Таким образом будет выделяться строка, которая будет выполнена при следующем шаге отладки. На текущий момент мы только запустили отладчик, первую строку программы он еще не выполнил, а только собирается выполнить. В левом нижнем окне переключитесь с активного окна Search на окно Stack Data. В этом окне мы сможем просматривать значения переменных

-Теперь выполним первую команду программы (`total = 0`). Для этого выберите команду Step Over. После выполнения этой команды первая строка программы выполнилась. Посмотрите в окно Stack Data. Там в разделе Local появилась переменная `total`, которой присвоено значение 0

-Теперь будем выполнять строчку `price = float(input())`, в которой требуется вводить данные. Выполните команду Step Over. Перейдите в окно Debug I/O и введите любое число, например, 45. Ввод завершите клавишей Enter. Пока программа ждала от нас ввода данных, окно Stack Data было неактивно и в окне редактора не было розовой подсветки исполняемой строки. После ввода данных и нажатия Enter окно Stack Data стало снова активным, в нем появилась переменная `price` с заданным нами значением. Значение переменной `total` пока не изменилось

-Выполним очередной шаг отладки (Step Over) и увидим, что значение переменной `total` изменилось и стало равно значению переменной `price`

Теперь самостоятельно продолжите отладку программы, выполняя команду Step Over, когда необходимо вводя значения и наблюдая, как меняются значения переменных в окне Stack Data.

Вы только что выполнили свою первую отладку программы!

3.Точка остановки

А теперь при помощи отладчика поищем ошибки в другой программе. Рассмотрим вот такую программу под названием bank:

```
print('Добро пожаловать в интернет-банк!')
print('У нас фантастические процентные
ставки!')
print('Для вкладов до 10 тысяч 
включительно прибыль составит 10%,')
print('для вкладов на сумму до 100 тысяч
включительно - 20%,')
print('для более 100 тысяч - 30%!')
print('На какую сумму желаете сделать
вклад?')
money = float(input())
if money <= 10000:
    money *= 1.1
if money <= 100000:
    money *= 1.2
if money > 100000:
    money *= 1.3
print('Вы получаете', money, ', поздравляем!')
```

Эта программа верно выводит 1 300 000, если ввести 1 000 000, и верно выводит 24 000, если ввести 20 000. Но, если ввести 1000, программа выведет 1320 вместо 1100, а если ввести 100 000 — выведет 156 000 вместо 120 000!

Выясним причины такого поведения при помощи инструментов отладки из меню Debug. Посмотрим, какие команды выполняются и как меняется значение переменной `money`.

Но если мы посмотрим на программу — начальные строчки нет смысла отлаживать пошагово — в них выполняется вывод на экран. Есть смысл делать отладку только с момента получения значения `money`.

Точка остановки

Если в строку, с которой начинается пошаговая отладка, поставить специальную метку — **точку остановки (breakpoint)**, тогда при пошаговом выполнении можно выполнить все команды до breakpoint'а в обычном режиме, а не пошагово.

Итак, отлаживаем программу bank.

-С помощью команды Add Breakpoint добавим точку остановки (breakpoint) на строчке 8, поместив туда курсор. Обратите внимание: тот же пункт меню теперь называется Remove Breakpoint (но только пока курсор на этой строчке кода), а перед строчкой появилась красная точка. Это индикатор breakpoint'a. Можно убрать breakpoint, щелкнув мышкой по красной точке, или установить его, щелкнув по этому месту на этой или на другой строчке.

-Запустим программу, но не кнопкой Run, а кнопкой Debug. В окне Debug I/O выведется текст из всех функций `print` и будет мигать курсор, ожидая ввода данных. Введем сумму — например, 1000. В этот момент программа доходит до строчки, на которой установлен breakpoint, и временно прерывает работу, отметив красным строку, на которой остановилась. В нижнем левом углу во вкладке Stack Data можно увидеть список переменных и их значения. Большинство — специальные системные, но в конце списка есть и переменная `money`.

-Если теперь выбрать в меню Start / Continue, программа продолжит работу до следующего breakpoint'a (которые можно ставить и снимать прямо во время отладки) или до конца. Но нам нужно пройти по программе построчно, чтобы понять, какие команды выполняются и чему при этом равно значение money. Чтобы продолжить выполнение программы, выберем команду Step Over.

-Продолжаем нажимать Step Over. Обратите внимание: когда программа выполнила строчку 9, значение `money` изменилось на 1100 — это верный итоговый ответ.

-Но что же дальше? После строчки 10, на которой программа проверяет условие `if`, выполняется строчка 11, после которой значение `money` увеличивается до неправильного! Неудивительно: ведь условие и правда истинно. Но программа явно должна работать по-другому

Исправьте программу так, чтобы она работала правильно.

На прошлом уроке мы смотрели, как работает цикл на примере трассировочных таблиц. А теперь давайте посмотрим в режиме отладки, как будет работать программа при вводе

последовательности 12, 3, 32, 14, 0.

```
count = 0
number = int(input())
while number != 0:
    if number % 10 == 2 and number % 4 == 0:
        count += 1
    number = int(input())
print('К о л и ч е с т в о  и с к о м ы х  ч и с е л:', count)
```

Пройдитесь по программе отладчиком, вводя указанные значения. Согласитесь, что так отлаживать программы проще, чем делать это вручную на бумаге.

На прошлом уроке мы познакомились с **алгоритмом Евклида**.

А теперь выберите разные значения для переменных a и b и пройдитесь отладчиком по данному коду:

```
a = int(input())
b = int(input())
while a != b:
    if a > b:
        a -= b
    else:
        b -= a
print(a)
```

Посмотрите, как будут меняться переменные. Определите, что будет выводиться на экран при выбранных вами значениях.

А теперь запишите модификацию алгоритма через деление. Какая программа быстрее находит результат?

Рассмотрим очередную задачу.

В программу поступают значения с температурного датчика. Ввод значений прекращается сигналом остановки (число 0).

Вот код этой программы:

```
a = int(input())

k = 0

t = a

while a != 0:
```

```
    if a > t:
        t = a
        k = 0
    if a == t:
        k += 1
    a = int(input())
print(k)
```

Ваша задача — определить, что находит и выводит данная программа относительно вводимых значений температуры?

При решении задач («Псевдонимы», «Лабиринт», «Бот-говорилка») помните, что они рассчитаны на людей. Пользователь должен понимать, что и когда нужно вводить.