Урок №4. Отладчик

План урока

1. Отладка и отладчик.
2. Команды отладчика в Wing IDE.
3. Пример отладки программы.

Аннотация

В уроке рассматривается работа с отладчиком на примере программы, которую нужно исправить. Затем даются группы задач, алгоритмы решения которых требуют рассмотрения многих случаев.

Процесс изучения и исправления ошибок в работе программы называется **отладкой**, по-английски — **debug**. 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, то сколько раз?

Для ответа на такие вопросы существуют специальные инструменты отладки (**отладчики**). Они позволяют проходить программу пошагово и следить при этом за значениями переменных. При этом проходить каждую строку обычно слишком трудоёмко (например, незачем заходить в какой-нибудь длинный цикл, если нас интересует другое место программы). Поэтому в программе можно установить одну или несколько точек остановки (breakpoint) — тогда при пошаговом выполнении можно выполнить все команды до **breakpoint'a** в обычном режиме, а не пошагово.

Средства отладки есть в стандартной библиотеке функций Питона, но обычно удобнее пользоваться возможностями среды программирования. Рассмотрим возможности отладчика Wing IDE на примере такой программы 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 `1300000`, если ввести `1000000`, и верно выводит `24000`, если ввести `20000`. Но если ввести `1000`, то программа выведет `1320` вместо `1100`, а если ввести `100000`, то она выведет `156000` вместо `120000`!

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

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

- Start / Continue и Stop Debugging. Кнопки Debug и Stop выполняют те же функции. (Если вы не видите подписей под кнопками их можно включить: Edit→Preferences...→User Interface→Toolbar→Toolbar Style→Icon and Text Below.) Первая команда запускает выполнение программы до ближайшего breakpoint'a, 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 одноимённая кнопка) доступен, когда программа запущена, но остановлена по breakpoint'y. Нажатие этой кнопки выполняет команду, записанную на текущей строчке, переходит на следующую и останавливается, ожидая ещё 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.

Итак, отлаживаем программу **bank**.

- С помощью команды Add Breakpoint добавим точку остановки (breakpoint) на строчке 8, поместив туда курсор — вряд ли ошибки есть раньше. 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, щёлкнув мышкой по красной точке, или установить его, щёлкнув по этому месту на этой или на другой строчке.
- Запустим программу, но не кнопкой Run, а кнопкой Debug. Как обычно, введём 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'a, выполняется строчка 11, после которой значение money увеличивается до неправильного! Неудивительно: ведь условие и правда истинно. Но программа явно должна работать по-другому.

Задача «Банк» (https://lms.yandexlyceum.ru/task/view/733) + исправьте программу так, чтобы она работала правильно.

Задача «Фибоначчи» (https://lms.yandexlyceum.ru/task/view/734).

Задача «Ним-пасьянс» (https://lms.yandexlyceum.ru/task/view/735).

Задача «Псевдоним-пасьянс» (https://lms.yandexlyceum.ru/task/view/736).

Задача «Псевдоним» (https://lms.yandexlyceum.ru/task/view/737) +

Задача «Ним 2 — пасьянс» (https://lms.yandexlyceum.ru/task/view/738).

Необязательные задачи:

- **«Ним 2» (https://lms.yandexlyceum.ru/task/view/740) +**
- **«Псевдоним-мизер» (https://lms.yandexlyceum.ru/task/view/741) +**
- **«Ним 3 — пасьянс» (https://lms.yandexlyceum.ru/task/view/742) +**
- **«Ним 3» (https://lms.yandexlyceum.ru/task/view/743) +**
- **«Лабиринт 2» (https://lms.yandexlyceum.ru/task/view/744) +**
- **«Бот-говорилка» (https://lms.yandexlyceum.ru/task/view/745) +**

Помните, что эти программы рассчитаны на людей. Пользователь должен понимать, что и когда нужно вводить.

Домашняя задача «Слова и буквы» (https://lms.yandexlyceum.ru/task/view/739).

В вашем распоряжении даже имеется решение этой задачи:

```
word = input()
shortest_word = word  # пока слово всего одно, оно и длиннейшее, и кратчайшее
longest_word = word
while word != 'стоп':
    word = input()
    word_length = len(word)
    if word_length < len(shortest_word):
        shortest_word = word
    if word_length > len(longest_word):
        longest_word = word
bad_letters = 0  # посчитаем, сколько букв отсутствуют в длинном слове
letter_number = 0
while letter_number < word_length:
    if shortest_word[letter_number] not in longest_word:
        bad_letters += 1
    letter_number += 1
if bad_letters == 0:
    print('ДА')
else:  # если нашлась хоть одна - ответ НЕТ
    print('НЕТ')
```

К сожалению, это решение неверное: оно правильно работает на приведённых в условии примерах, но не проходит дальнейшие тесты. Попробуйте определить, что не так в этом решении, и исправить его.