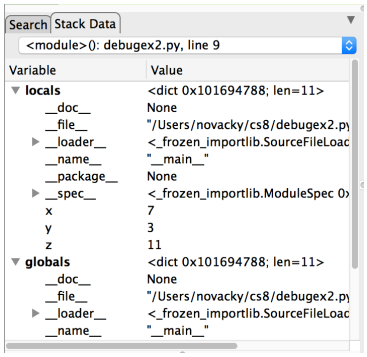
Wing 101's Debugger

A breakpoint is an intentional stopping or pausing place in a program for debugging purposes. To place a breakpoint in a program, click the mouse to the right of the line number in the darker green margin on any line of code. A red dot appears indicating that a breakpoint is set. To remove this breakpoint, click the red dot with the mouse. Multiple breakpoints may be set at various places in a program at any time. In the program on right, a breakpoint was set at line 9. Breakpoints can also be set or removed using the Debug menu.

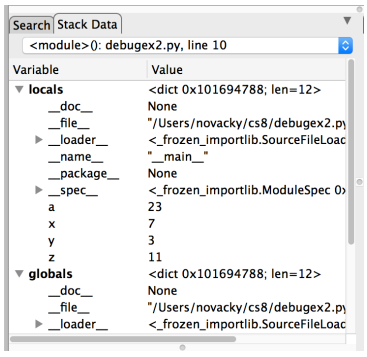
```
1  def f(u,v):
2      k = u - v
3      return k**2
4
5
6  x = 7
7  y = 3
8  z = 11
9  a = 2*x + 3*y
10 b = f(a,y) + x
11
```

Now run this program by clicking the "bug" icon on the menu bar shown here. The program executes through the line immediately before the breakpoint (through line 8). The **Stack Data** tab, shows the state of the program after the first eight lines have been executed. Note that variables **x**, **y**, and **z** are listed as local variables and their values are 7, 3, and 11 respectively. In addition, the variable **a** is not shown, since line 9 has not been executed yet.



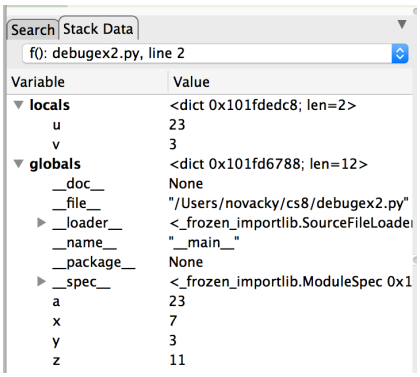
The (rightmost) three buttons on the menu bar are called **Step Into (F7)**, **Step Over (F6)**, and **Step Out (F8)** respectively. **Step Over** is used to execute one line of code or to execute every line of code inside a function and return to the caller. **Step Into** is used to enter a function and stops at the first executable statement (from here it is possible to execute one statement at a time by using **Step Over** repeatedly). The **Step Out** button causes the rest of the function to execute and return to the calling program.

In order to execute line 9, click the **Step Over** button. Variable **a** is now assigned the integer value 23. See the **Stack Data** tab below.



At this point, the first nine lines have been executed. If we choose to click **Step Over,** line 10 will be executed immediately and the program's execution is complete. However, doing this doesn't give us a chance to examine the code in the function call **f(a, y)**. Instead, click **Step Into**, this takes us to the first statement (line 2) of function f() (see below). Here we take a look at the Stack Data. Note that the formal arguments **u** and **v** of the function call f(a, y) are shown with **u** = 23 and **v** = 3. So variables **u** and **v** point to **a** and **y** respectively.

```
1  def f(u,v):
2      k = u - v
3      return k**2
4
5
6  x = 7
7  y = 3
8  z = 11
9  a = 2*x + 3*y
10 b = f(a,y) + x
11
```



Debugging User-defined Classes

Here is the definition of the BritishLength class:

```
class BritishLength:

    def __init__(self, feet, inches):
        self.__feet = feet + inches // 12
        self.__inches = inches % 12

    def __str__(self):
        return '%d"%d\'' % (self.__feet, self.__inches)


    def toInches(self):
        return 12 * self.__feet + self.__inches


    def __add__(self,other):
        feet = self.__feet + other.__feet + (self.__inches + other.__inches) // 12
        inches = (self.__inches + other.__inches) % 12

        return BritishLength(feet,inches)


    def feet(self):
        return self.__feet

    def inches(self):
        return self.__inches
```

Let's use Wing 101's debugger to look at a BritishLength object.

Debugging Objects with Wing 101

In the code below, a breakpoint is set on line 4. At this point in the program variable **bl1** has been initialized to 6'4".

```
1  from BritishLength import *
2
3  bl1 = BritishLength(6,4)
4  print('bl1 =  ',bl1)
5  bl2 = BritishLength(3,9)
6  print('bl2 = ',bl2)
7
8  print('bl1 to inches = ', bl1.toInches())
9  print('bl2 to inches = ', bl2.toInches())
10
11 print('The sum of bl1 and bl2 = ', bl1 + bl2)
12
13
```

Let's examine the contents of variable **bl1**, click on the triangle on the left to see the details.

Note the data components of variable bl1 are 6 and 4 respectively.