Busy bees

General information

Infinite loops

Take care to avoid infinite loops. An infinite loop is a loop that never stops executing: in most of the cases it concerns a `while`-loop where the statements inside the loop never take care to make the `while`-condition `False` after some time. As an example, take a look at the following code snippet

```
>>> i = 0
>>> a = 0
>>> while i < 4:
...     a += 1
```

Because the statement `a += 1` will never cause the initial value of the variable `i` to become larger than or equal to 4, the condition `i < 4` will evaluate to `True` forever.

Tip: If you work with Eclipse, you can check whether a program that was started is still running, if you observe a red square in the top menu of the Console. If you click the red square, you force the program to stop.

Exchange the value of two variables

In Python you can exchange the values of two variables by using a third variable as a temporary reference.

```
>>> x = 5
>>> y = 3

>>> temp = x
>>> x = y
>>> y = temp

>>> print(x)
3
>>> print(y)
5
```

However, this can be done using a shorthand notation that makes use of the fact that an assignment statement first evaluates the right-hand side (where you can fetch the old values of the variables), and only than assigns the result to the variables in the left-hand side. As a result, you can exchange the value of two variable without the need to use a temporary variable.

```
>>> x = 5
>>> y = 3

>>> x, y = y, x

>>> print(x)
3
>>> print(y)
5
```

As soon as you have learned how to work with tuples in Python, you'll call this a *tuple assignment*. Take not that the following code fragment illustrates how the exchange of the value of two variables should **not** be implemented.

```
>>> x = 5
>>> y = 3

>>> x = y
>>> y = x

>>> print(x)
3
>>> print(y)
3
```

At the time the assignment statement `y = x` is executed, both variables `x` and `y` already reference the same value (the integer 3), and the reference to the integer 5 has already been lost.

Remarks

In case you setup Eclipse to make use of pylint (Window > Preferences > PyDev > Pylint > Use pylint?), additional markers are placed in the right margin of the code editor that indicate bad programming style. One type of markers indicates that variables have been defined that are not used in the source code. This does not make your Python code invalid, but may cause this variable to accidentally be misused in your code. Therefore it is considered bad programming style.

The above sitation may occur, for example, when you combine a `for`-loop in combination with the `range` function to execute a sequence of statements a fixed number of times. Because the syntax of the `for`-loop requires you to always use a loop variabel, you may have to use a variable that is not necessarily used in your source code. If this is the case, it is commonplace to use an underscore (`_`) as the variable name, which implicitly expresses in the Python community that *this is a variable that will not be used in the source code*. Pylint will take this into account, and will never indicate that an underscore is a variable that is not used in the code. The code snippet

```
>>> count = 0
>>> for i in range(4):
...     count = 2 * count + 1
...
```

may thus better be rewritten as

```
>>> count = 0
>>> for _ in range(4):
...     count = 2 * count + 1
...
```

Specific information

The evolution of the number of male bees, the number of female bees and the total number of bees from one generation to another can be represented graphically in the following way

Herewith, the number of male bees in generation i equals the number of female bees in generation $i-1$ (the previous generation). As a result, the number of female bees in generation i equals the sum of the number of male and female bees in generation $i-1$, because the number of male bees in that generation equals the number of female bees in generation $i-2$.

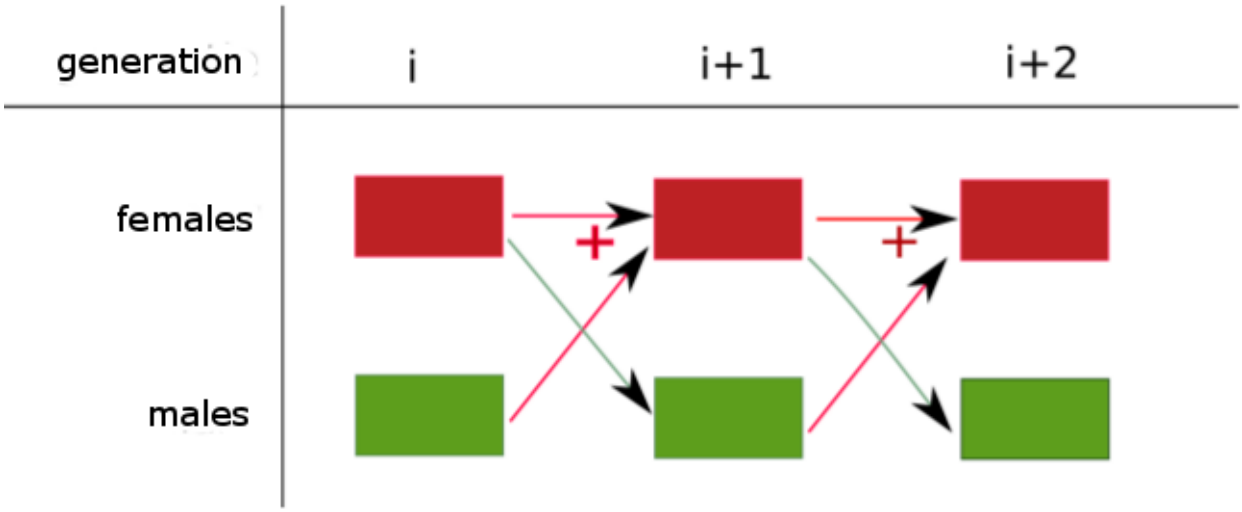

Figure 1: Busy bees

Canvascrack

General information

Counting starts at zero

Computer scientists by default start counting from zero, not from one, and Python follows this tradition in many of its design decisions. As an example, the built-in function `range` generates a sequences of successive integers that starts at zero, if you only pass a single argument to the function. Here's how you count to 5 in Python

```
>>> for i in range(6):
...     print(i)
...
0
1
2
3
4
5
```

If you want counting to start at another value, you can pass this value as an extra argument to the `range` function.

```
>>> for i in range(1, 6):
...     print(i)
...
1
2
3
4
5
```

However, it is considered a more *Pythonic* solution to write the above as

```
>>> for i in range(5):
...     print(i + 1)
...
1
2
3
4
5
```

Possible errors

Python uses the ASCII character set by default, which only contains 256 different characters. ASCII only supports a limited amount of accented characters, which makes that characters such as ö or ë are illigal. If you use non-supported characters in your source code, you may see the following error message.

```
SyntaxError: Non-UTF-8 code
```

This can easily be resolved by setting up Eclipse to support a more extensive character set. The default character set that is used nowadays is UTF-8. You can setup Eclipse to use this character set, by executing the following steps:

- go to the menu `Window` > `Preferences`
- chose `General` > `Workspace`
- under `Text file encoding` (bottom of the dialog) chose `Other`: `UTF-8`

For a more detailed explanation about changing the character set used by Python, and alternative ways to the above procedure (which is the easiest one), we refer to the document "Getting Started with Python" that is linked as a manual in the overview of the first series of exercises on the Pythia platform.

Erdős-Straus conjecture

Specific information

In this assignment you might be tempted to compute the value z if the values x, y and n are known. After all, it holds that

$$z = \left(\frac{4}{n} - \frac{1}{x} - \frac{1}{y}\right)^{-1}$$

However, we strongly advise to make explicit use of this formula in your source code, because it uses floating point divisions to compute z that might cause rounding errors, leading to wrong results. As an example, consider the computation in the following code snippet (the exact value of z should be 12).

```
>>> n, x, y = 4, 1, 4

>>> z = 1 / (4 / n - 1 / x - 1 / y)
>>> z
12.000000000000001
```

With larger integers, the rounding errors grow worse. There exist numerical techniques to keep the rounding errors of z under control, but these are quite complicated and are beyond the scope of an initial programming course.

A Martian census

General information

Premature abortion of loops

In Python you can use the statements `break` and `continue` to abort a loop before it has come to completion. In general, however, these statements are considered to give a bad programming style.

One situation where you may want a premature abortion of a loop occurs when you want to find a solution by trying all possible cases, and stop as soon as one solution has been found. Instead of using `break` or `continue` in this case, it is better to use an additional Boolean variable that indicates whether or not the solution has already been found.

```
>>> found = False
>>> while not found:
...     if (solution found):
...         found = True
...
```

As soon as the solution has been found (represented here by the fact that the condition *solution found* evaluates to `True`), the variable `found` is assigned the value `True`. As a result, the `while`-loop ends the next time the `while`-conditions is evaluated after the current iteration.

Conversion of values to Boolean values

In Python it is generally considered a better programming style (more *pythonic*) to rewrite the condition in the following code snippet

```
if x != 0:
    pass
```

in short as

```
if x:
    pass
```

This is possible, because the evaluation of the condition in an `if` statement or a `while` statement, implicitly converts the expression into a Boolean value. For most data types, all values are converted to the Boolean value `True`, except for a single value that is converted to `False`:

- for integers only 0 is converted to `False`
- for floats only 0.0 is converted to `False`
- for strings only the empty string (`''`) is converted to `False`
- for lists only the empty list (`[]`) is converted to `False`
- ...

As a result, you will encounter this shorthand notation very often in code examples that you find in books or online. So, even if you find the longer notation more readable, it is still necessary to understand the shorthand notation when trying to understand code examples that make use of it.

Also not that it is quite useless to write

```
if found == True:
    pass
```

as the variable `found` already references a Boolean value. Also in this case, it is shorter to write

```
if found:
    pass
```