General

The PyDev plugin in Eclipse sometimes tries to be TOO smart, by automatically adding import statements to the source code. If this happens to import modules that are not part of The Python Standard Library, the Pythia environment will not recognise these modules and report an `internal error`.

To avoid this, you can deactivate the automatic addition of import statements to your source code, by unchecking the following setting:

`Window` > `Preferences` > `PyDev` > `Editor` > `Auto Imports` > uncheck all check boxes

Torn numbers

General information

Possible errors

Pay attention to the fact that the built-in function `int` cannot convert an empty string into an integer (try to think for yourself what integer this should be). If you try to do this anyway, you may get the following error message.

```
>>> int('')
Traceback (most recent call last):
ValueError: invalid literal for int() with base 10: ''
```

Word evolutions

General information

Remarks

The string method `index` may be used to determine the position of the leftmost occurrence of a character in a strings.

```
>>> string = 'mississippi'
>>> character = 'i'
>>> string.index(character)
1
```

This code fragment finds the leftmost occurrence of the letter `i` at position 1 in the string `mississippi`. Note that Python indexes the positions of the characters in a string starting from 0, not starting from 1.

It should be noted that this methode might be inefficient in case the position of the character can be derived directly. In order to find the position of a character that first occurs at position p, the index method must traverse the first $p - 1$ positions of the string. For a character that does not occur in the string, the string method index must even travers all characters in the string before it can decide that the character was not observed.

And now for something completely upside down

General information

Escaping literal backslashes

In Python a backslash is used inside a string to escape the next character in the string, so that this character loses its special meaning (for example to put a double quote inside a string that is enclosed itself in between a pair of double quotes) or to give a special meaning to the next character (such as the end of line that is represented by the character `'\n'` or a tab that is represented by the character `'\t'`).

Because this gives a special meaning to a backslash inside a string (used to escape characters), a literal backslash inside a string must be represented by two successive backslashes (`'\\'`), where the first backslash serves as the escape symbol and the second backslash is the character that is given its literal meaning by escaping it.

Representation of newlines

In Python you may represent the end of a line as the string `'\n'`. For example, if you want to construct a string that represents multiple lines of text (a *multiline string*), you may use the following strategy.

```
>>> text = 'Here is a first line'
>>> text += '\n'
>>> text += 'And a second line'
>>> text
'Here is a first line\nAnd a second line'
>>> print(text)
Here is a first line
And a second line
```

Read multiple lines from input

If you know in advance how many lines must be read from input, you may use the following strategy

```
>>> lines = 4
>>> for _ in range(lines):
...     line = input()
...     # process the line
...
```

If the number of lines that must be read from input is not known in advance, but you know for example that the last line is an empty line, you may use the following strategy

```
>>> line = input()
>>> while line:
...     # process the line
...     line = input()
...
```

Nob's number puzzle

General information

Remove leading and/or trailing whitespace

In Python you can use the string method `strip` to remove leading and trailing whitespace (spaces, tabs and newlines). In case you only want to remove leading whitespace, you can use the string method `lstrip`. In case you only want to remove trailing whitespace, you can use the string method `rstrip`. You can also pass an argument to these string methods, that indicates which leading and/or trailing characters have to be removed. For more details about these string methods, we refer to The Python Standard Library.

```
>>> text = '  This is a text  '
>>> text.strip()
'This is a text'
>>> text.lstrip()
'This is a text  '
>>> text.rstrip()
'  This is a text'
```

Align strings over a fixed number of positions

You can use the *format specifier* of the string method `format` to reserve a fixed number of positions to output a given text fragment. This is done by formatting the text as a string (indicated by the letter `s`), preceded by an integer that indicates the fixed number of positions that needs to be reserved for the string.

```
>>> '{:5s}'.format('abc')   # reserve 5 positions
'abc  '
```

In case the string is longer than the number of reserved positions, the placeholder is replaced by the entire string, and thus takes more than the reserved space. In case the string is shorter than the number of reserved positions, the string is left aligned by default. You can also explicitly define the type of alignment in the *format specifier*: left, right or centered. Python will automatically pad the string with additional spaces to the left and/or to the right.

```
>>> '{:<5s}'.format('abc')   # reserve 5 positions, left alignment
'abc  '
>>> '{:>5s}'.format('abc')   # reserve 5 positions, right alignment
'  abc'
>>> '{:^5s}'.format('abc')   # reserve 5 positions, centered alignment
' abc '
```

In case a centered alignment is chosen and the number of additional spaces is an odd number, Python prefers to add one more space to the right than to the left.

Remarks

To repeat a given string a fixed number of times, you can multiply that string with an integer. As with the multiplication of numbers, this uses the `*` operator. The order of the string and the integer does not matter in the multiplication

```
>>> 'a' * 3
'aaa'
>>> 5 * 'ab'
'ababababab'
```

This is very handy in case the number of repetitions of the string is not known beforehand, but for example can be retrieved from a variable.

```
>>> repetitions = 4
>>> repetitions * 'abc'
'abcabcabcabc'
```

Bible codes

Specific information

In order to read the word that is hidden in a Bible verse, we advise you to use the following strategy. As a first step, you may read all lines of the entire verse from input, and convert those to a single string that only contains the letters from the Bible verse. Indeed, all other characters must be ignored. For example, if you get the following Bible verse that is spread over two lines

```
And hast not suffered me to kiss my sons and my daughters?
thou hast now done foolishly in so doing.
```

Then, based on the first step, you only keep the successive letters in the Bible verse.

```
Andhastnotsufferedmetokissmysonsandmydaughtersthouhastnowdonefoolishlyinsodoing
```

After you have obtained this string of letters, you may read the hidden word by starting with the letter at the given position, and read the next letters by jumping the given number of letters forward or backward. For example, if you start reading from the 45th letter in the above string, and repeatedly jump 4 letters forward into the string until you have read a 7-letter word, than you obtain the word *roswell*.