General

Assignment vs. equality test

In Python the syntax of an assignment statement uses a single equal sign, where an equality test (check whether two objects have the same value) uses two successive equal signs. To check if the value of the variable `x` equals the integer 2, you write

```
>>> if x == 2:    # correct
...     pass
```

and not

```
>>> if x = 2:     # wrong
  File "<myscript.py>", line 1
    if x = 2:
         ^
SyntaxError: invalid syntax
```

Personal warmth

General information

Remarks

Accurate definition of the number e

An accurate definition of the number e can be found in the `math` module.

```
>>> import math
>>> math.e
2.718281828459045
```

Birthstones

General information

Avoid multiple `print` statements

It is always a good idea to avoid excessive use of `print` statements, in order not to clutter your source code. As such, it becomes much easier to track what exactly will be printed and how the output is formatted.

Say, for example, that you have the following source code

```
>>> x = 3
>>> if x < 5:
...     print('less than 5')
... else:
...     print('more than 5')
...
'less than 5'
```

It is better to rewrite this source code as

```
>>> x = 3
>>> if x < 5:
...     result = 'less'
... else:
...     result = 'more'
...
>>> print('{} than 5'.format(result))
'less than 5'
```

Knight move

General information

Split string in its individual characters

If you want to assign the n characters in a string to n individual variables, you can use the following trick.

```
>>> word = 'ab'
>>> first, second = word
>>> first
'a'
>>> second
'b'
>>>
>>> woord = 'abcd'
>>> first, second, third, fourth = word
>>> first
'a'
>>> second
'b'
>>> third
'c'
>>> fourth
'd'
```

This is an example of a more generic technique called *tuple unpacking*.

Convert letters to their ordinal value

In Python, each character has a corresponding ordinal value (an integer). For example, the question mark (`?`) has ordinal value 63. The value can be easily obtained using the built-in function `ord`.

```
>>> ord('?')
69
```

The interesting thing about these ordinal values, is that successive letters in the alphabet have successive numerical values. This holds for both uppercase and lowercase letters. If we know that the letter `a` has ordinal value 97, it immediately follows that the letter `b` must have ordinal value 98. With this knowledge, we can easily determine the position of a letter in the alphabet.

```
>>> letter = 'd'
>>> ord(letter)
100
>>> ord('a')
97
>>> ord(letter) - ord('a') + 1   # d is the 4th letter of the alphabet
4
>>> ord('z') - ord('a') + 1      # z is the 26st letter of the alphabet
26
```