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
```

Reynolds number

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('f{result} than 5')
'less than 5'
```

Rock-paper-scissors

Conditional expressions

You can use a conditional statement, if you want the value that must be assigned to a variable to depend on a certain condition. Say, for example, that you work with the variable `x` that refers to the length of a person (in centimeters), and that you want to assign the value `'large'` to the variable `length` if $x > 185$, and the value `'small'` otherwise. Using a conditional statement, this can be done in the following way.

```
>>> x = 100
>>> if x > 185:
```

```
...     length = 'large'
... else:
...     length = 'small'
...
>>> length
'large'

>>> x = 190
>>> if x > 185:
...     length = 'large'
... else:
...     length = 'small'
...
>>> length
'large'
```

The same result can be obtained using an `if-else`-expression (also called a *conditional expression*). In contrast with a conditional statement, this is an expression (that evaluates into a value) and not a statement. As a result, you may use conditional expressions in the right-hand side of an assignment statement. Using conditional expressions, the above interactive session can be written much shorter.

```
>>> x = 100
>>> length = 'large' if x > 185 else 'small'
>>> length
'small'

>>> x = 190
>>> length = 'large' if x > 185 else 'small'
>>> length
'large'
```

Seasons

Testing if the value of a variable belongs to a fixed set of options

The following conditional statements have a logical error in checking whether or not it is weekend or a working day on a given weekday. Actually, the condition will always be `True`, suggesting that it is forever weekend.

```
>>> weekday = 'sunday'
>>> # WRONG!!
>>> 'weekend' if (weekday == 'saturday' or 'sunday') else 'working day'
'weekend'

>>> weekday = 'monday'
>>> # WRONG!!
>>> 'weekend' if (weekday == 'saturday' or 'sunday') else 'working day'
'weekend'
```

The reason things go wrong is that the condition is composed out of two smaller conditions, being `weekday == 'saturday'` and `'sunday'`. The second condition is always true, since any string is `True` except for the empty string which is `False`. The correct way of formulating the composed condition is

```
>>> weekday = 'sunday'
>>> 'weekend' if (weekday == 'saturday' or weekday == 'sunday') else 'working day'
'weekend'
```

```
>>> weekday = 'monday'
>>> 'weekend' if (weekday == 'saturday' or weekday == 'sunday') else 'working day'
'working day'
```

Note the repetition of the variable name in the condition.

Trilateration

Check if a number is between two limits

In Python you can check if the value (of a variable) is within a certain interval by using a composite Boolean expression. For example, if you want to check whether or not the number `x` is in the interval $]a, b[$, you can use the following Boolean expression

```
>>> a < x and x < b
```

In mathematics, this condition would be written as $a < x < b$, and Python can use the same kind of shorthand notation.

```
>>> a < x < b:
```

This clearly shows that Guido Van Rossum, who invented the Python programming language, was a trained mathematician. Equally, the condition $a \leq x \leq b$ where the limits are included in the interval can be written using the same shorthand notation in Python.

```
>>> a <= x <= b:
```

Comparing two floating point numbers

Computers cannot always exactly represent floating point numbers. As a result, we always have to take into account small rounding errors while computing with floating point numbers.

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
>>> 1 / 3 == 5 / 6 - 3 / 6
False
```

Therefore, instead of using the comparison operator `==` to check if two floating point numbers are equal, we could say that two floating point numbers are equal if the do not differ more than a very small value ϵ (e.g. 10^{-6}).

```
>>> a = 1 / 3
>>> b = 5 / 6 - 3 / 6
>>> epsilon = 10 ** -6
>>> a - epsilon < b < a + epsilon
True
```

The latter can be rewritten as

```
>>> abs(a - b) < epsilon
True
```

where we say that the difference between the two values is not greater than the rounding error ϵ.

Darts

Accurate definition of the number π

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

```
>>> import math
>>> math.pi
3.141592653589793
```

Cyclometric functions from the `math` module

The `math` module from the Python Standard Library defines a couple of **cyclometric functions** (or inverse trigonometric functions) such as the arcsine function (`asin`), the arccosine function (`acos`) and the arctangent function (`atan` or `atan2`; the main difference between these two is that the function `atan2` takes into account the quadrant in which the point is located). The domain of the arcsine and the arccosine functions is $[-1, 1]$. This means that these function only take *floating point* values from the interval $[-1, 1]$. As a result, you must take care that rouding errors do not cause values outside this domain. The following example illustrates how the detrimental effect of rounding errors can be remedied.

```
>>> import math
>>> value = 1.00001
>>> math.acos(value)                    # compute arccosine
Traceback (most recent call last):
ValueError: math domain error
>>> value = max(-1.0, min(value, 1.0)) # guarantee that value is in interval [-1, 1]
>>> value
1.0
>>> math.acos(value)
0.0
```