Personal warmth

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
```

Mondrian

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:
```

Counterfeiting

Specific information

For both the first and second weighing, we can respectively determine the group m and the item in the group n that is lighter than the others. This way, we can determine the counterfeit coin as the coin with number $3m + n + 1$.

If the counterfeit coin is in the first group (group 1-2-3) during the first weighing, m will be equal to 0. If the counterfeit coin is in the second group (group 4-5-6), then m is equal to 1 and if it is in the third group (group 7-8-9) then m is equal to 2.

The second weighing determines the item n in exactly the same way.

Knight move

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
```

General

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.

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'
```

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'
```