General

Formatted text: string method `format`

When you want to compose a string in a certain way from fixed and variable fragments, it may turn out handy to make use of the string method `format`. This method can be called on any string that serves as a template. Each variable fragment in the template is indicated with a pair of curly braces (`{}`). For each variable fragment in the template string, an argument must be passed to the `format` method. The methode will fill up the value of each argument into the corresponding position in the template string, so that a new (filled-up) string is formed that is returned by the method.

The following code fragment defines three variables `x`, `y` and `sum`, with `sum` being calculated as the sum of the variables `x` and `y`. The code makes use of the string method `format` to compose a string that is formatted as `x + y = sum`, with `x`, `y` and `sum` being filled up with the values of the variables having the same name. The string that is returned by the string method `format` is then printed using the built-in function `print`.

```
>>> x = 2
>>> y = 3
>>> sum = x + y
>>> print('The sum of {} and {} is {}'.format(x, y, sum))
The sum of 2 and 3 is 5
```

A pair of curly braces in a template string is often called a *placeholder*. By default, the first placeholder is filled up with the value of the first argument passed to the string method `format`, the second placeholder with the value of the second argument, and so on. If this is the case, we say that the placeholders are filled up *positionally*. The string method `format` supports other ways to fill up the placeholders, and supports a more detailed specification of how the values are formatted when they fill up the placeholders. We refer to The Python Standard Library for more details about the string method `format`.

Output floats with a fixed number of decimal digits (rounded)

By default the built-in function `print` format floating point numbers using a large number of decimal digits. However, sometimes you will want to print floating point numbers with a fixed number of decimal this. You could try to use the built-in function `round` to achieve this, as it allows rouning of numbers up to a given number of decimal digits.

```
>>> print(1 / 3)
0.3333333333333333
>>> print(round(1 / 3, 2))
0.33
```

The problem with this solution is that rounding errors due to the internal representation of floating point numbers, may generate numbers that are not printed with the desired number of decimal digits.

A better solution makes use of the string method `format` to specify the number of decimal digits when formatting floating point numbers as text. Inside a pair of curly braces that represents a placeholder in the template string, you may specify how the value that fills up the placeholder must be formatted. This is done by placing a so-called *format specifier* in between the curly braces. The format specifier itself is preceded by a colon (`:`).

To format a value as a floating point number with a fixed number of decimal digits, you can use the format specifier `:.nf. Here, the letterf‘` indicates that the value must be formatted as a floating point number, and the number n indicates the number of decimal digits. The following code shows, for example, how a number can be formatted as a floating point number, rounded up to two decimal digits.

```
>>> print('{:.2f}'.format(1 / 3))
0.33
```

We refer to The Python Standard Library for more details about the use of *format specifiers*.

Remainder after integer division: the module operator (%)

In Python you can use the modulo operator (%) to determine the remainder after integer division. If both operandi are integers, the result is itself an integer. As soon as one of the operandi is a float, the result will be a float.

```
>>> 83 % 10
3
>>> 83.0 % 10
3.0
>>> 83 % 10.0
3.0
>>> 83.0 % 10.0
3.0
```

Extra mathematical functionality: the `math` module

It is an explicit design choice to keep the Python programming language as small as possible. However, there are mechanism built into the language to extend the language with new functionality. When Python is installed, a number of these modules are shipped with it. These modules together are referred to as The Python Standard Library.

The `math` module is one of these methods from the The Python Standard Library. As you might derive from its name, the `math` module adds some mathematical functionality to Python. Before you can start using this functionality, however, you must first import the module. There are two ways in which this can be done.

The first way imports the module as a whole. After this has been done, you must prefix the names of variables, functions or classes that are defined in the module with the name of the module and a dot if you want to use them in your Python code.

```
>>> import math
>>> math.sqrt(16)          # square root
4.0
>>> math.log(100)          # natural logarithm
4.605170185988092
>>> math.log(100, 10)      # log10
2.0
>>> math.pi                # accurate value of pi
3.141592653589793
```

The second way only imports some specific names of variables, functions or classes in your Python code. After this has been done, you can directly use these names without prefixing them.

```
>>> from math import sqrt, log, pi
>>> sqrt(16)               # square root
4.0
>>> log(100)               # natural logarithm
```

```
4.605170185988092
>>> log(100, 10)          # log10
2.0
>>> pi                    # naccurate value of pi
3.141592653589793
```

We refer to The Python Standard Library for a complete overview of the variables and functions defined in the `math` module.

Floating point division versus integer division

Python makes a clear distinction between floating point division (indicated by the operator `/`) and integer division (indicated by the operator `//`). Floating point division always results in a float. However, with integer division, the data type of the result depends on the data type of the operandi. If both operandi are integers, the result is an integer as well. If one or two of the operandi are floats, the result is itself a float.

```
>>> x = 8
>>> y = 3
>>> z = 4
>>> x / y            # floating point division of two integers
2.6666666666666665
>>> x // y           # integer division of two integers
2
>>> float(x) // y    # integer division of a float and an integer
2.0
>>> x / z            # floating point division of two integers
2.0
>>> x // z           # integer division of two integers
2
```

Python decides which kind of division to use solely based on the operator that is being used. The choice between floating point division or integer division is not influenced by the data types of the operandi.

```
>>> x = 7.3
>>> y = 2
>>> x // y
3.0
>>> y // x
0.0
>>> x / y
3.65
```

Heartbeats

General information

Remarks

Exponentiation in Python

Just as Python has operators for addition (`+`) and multiplication (`*`), it also has a power operator: `**`.

```
>>> 10 ** 2   # the square of 10
100
>>> 2 ** 3    # the cube of 2
8
```

The diatomist

Specific information

A circle with radius r has a circumference of length $2\pi r$ and encloses an area of size πr^2.

Timekeeping on Mars

General information

Use of the modulo operator and integer division for the conversion of units

Consider the integer variable `secs` that refers to an elapsed time in seconds. You can convert this time period to hours, minutes and seconds as follows.

```
>>> secs = 123456

>>> hrs = secs // 3600
>>> mins = (secs // 60) % 60
>>> secs = secs % 60

>>> print('{} hours, {} minutes, {} seconds'.format(hrs, mins, secs))
34 hours, 17 minutes, 36 seconds
```

Clock hands

General information

Add leading zeros using string method `format`

If you want to convert an integer to fixed-length string with leading zeros, you can use the string method `format` and an accompanying *format specifier*. Format specifiers are always put in between the pair of brackets used as a placeholder in the template string. A format specifier always starts with a colon (`:`).

In particular, to convert an integer into a fixed-length string with leading zero, you use the format specifier `:0nd`. In this format specifier, the `0` indicates leading zeros need to be added in case the string would otherwise be shorter than the target length, the letter `d` indicates that the value must be formatted as a digit (number) and the number `n` indicates the target length of the string. If the number you want to convert to a string is longer than the target length, the entire number is converted into a string. In this case, the resulting string will thus be longer than the target length.

```
>>> "{}".format(2)
'2'
>>> "{:02}".format(2)
'02d'
>>> "{:02d]".format(34)
'34'
>>> "{:02d}".format(567)
'567'
>>> "{:06d}".format(89)
'000089'
```

There are other ways to convert a number into a fixed-length string with leading zeros. One option is to use the string method `zfill`. You can also compute the difference between the actual length and the target length, to determine how many leading zeros must be prepended. Or you can use a `while` loop to prepend leading zeros until the target length is reached.

```
>>> target = 3
>>> number = str(2)
>>> number.zfill(target)
'002'
>>> leading_zeros = target - len(number)
>>> leading_zeros
2
>>> '0' * leading_zeros + number
'002'
>>> while len(number) < target:
...     number = '0' + number
...
>>> number
'002'
```

Remarks

Determine the smallest value

The built-in function `min` can be used to determin the minimum of two values.

```
>>> min(7, 3)
3
>>> min(3.14, 7.45)
3.14
```

The same function can also be used to determine the minimum of multiple values.

```
>>> min(7, 3, 8, 19, 2, 12)
2
>>> min(3.14, 7.45, 17.35, 373.21, 2.34, 98.36)
2.34
```

Absolute value

The built-in function `abs` can be used to compute the absolute value of a number.

```
>>> abs(42)
42
>>> abs(-42)
42
>>> abs(3.14159)
3.14159
>>> abs(-3.14159)
3.14159
```

Specific information

Each position of a clock hand can be expressed as the number of degrees of the angle it makes (clockwise, of course) with the direction of 12 o'clock. For example, at 3 the clock hand makes an angle of 90° with the direction of 12 o'clock.

The angle in between the two clock hands can then be computed as the difference between their positions expressed in degrees.

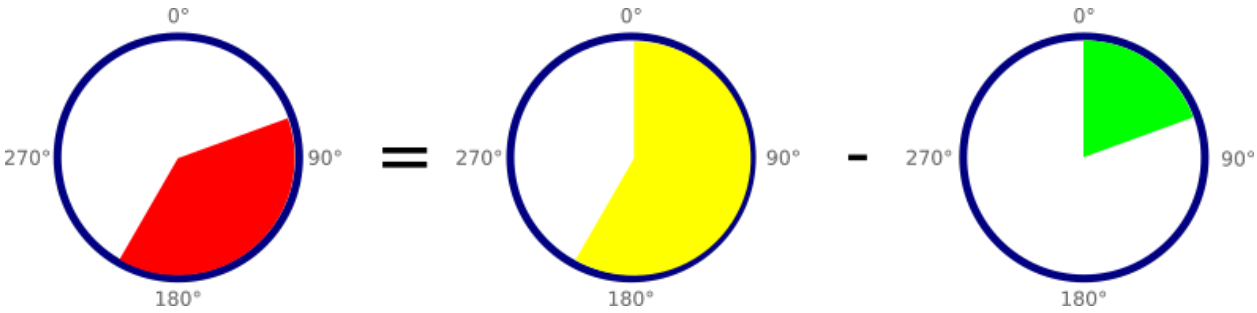

Figure 1: Angles