General

Formatted text: string interpolation

When you need a controlled way to compose a string as a mix of fixed and variable fragments, it might be handy to make use of string interpolation. An **interpolated string** is a regular string that is prefixed with the letter `f` (in front of the opening single or double quote). As a result, interpolated strings are also called **f-strings**.

An f-string serves as a kind of template, with each variable fragment indicated by a pair of curly braces (`{}`). In between these curly braces you place an expression whose value will fill up the position of the variable fragment in the resulting string.

For example, in the following code fragment we define two variables `number1` and `number2` whose sum we want to output. We use string interpolation to output formatted text that contains the two individual terms and the result of adding the two terms.

```
>>> number1 = 2
>>> number2 = 3
>>> print(f'The sum of {number1} and {number2} is {number1 + number2}.')
The sum of 2 and 3 is 5.
```

A pair of curly braces in an interpolated string is called a **placeholder**. Inside such a placeholder you cannot only put an expression, but after a colon you can also specify how the value of that expression must be formatted (read: how it needs to be converted into a fixed string). More details about the different ways to specify this formatting can be found in The Python Standard Library.

Mercator projection

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 selection of these modules are shipped as well. These modules are referred to as The Python Standard Library.

The `math` module is one of these modules 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.

Trigonometric functions from the `math` module

The `math` module from the Python Standard Library defines a couple of **trigonometric functions** such as the sine function (`sin`), the cosine function (`cos`) and the tangent function (`tan`). It's important to pay attention to the fact that these functions expect an angle expressed in radians, and not in degreees. Luckily enough, the `math` module also defines functions to convert an angle expressed in degrees into radians (`radians`) and *vice versa* (`degrees`).

```
>>> import math
>>> angle = 90
>>> radians = math.radians(angle)
>>> radians
1.5707963267948966
>>> radians == math.pi / 2
True
>>> math.cos(radians) # must evaluate to 0, but note the rounding error
6.123233995736766e-17
>>> math.sin(radians)
1.0
```

Vis viva

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` afloat‘.

```
>>> 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
```

Remainder after integer division: the modulo 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
```

Notation of floating point numbers

In Python, floating point numbers are written with a decimal dot, not with a comma. Commas are used by Python to separate the arguments of a function or the elements of a compound data type.

```
>>> 3.14159          # floating point number
3.14159
>>> 3,14159          # tuple of two integers
(3, 14159)
```

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
```

Square root

The square root of a number can be computed using the `sqrt` function from the `math` module.

```
>>> import math
>>> math.sqrt(121)
11.0
>>> math.sqrt(1234)
35.12833614050059
```

Because $\sqrt{x} = x^{1/2}$ the power operator (`**`) can be used as well to compute the square root.

```
>>> 121 ** (1 / 2)
11.0
>>> 1234 ** 0.5
35.12833614050059
```

The cubic, fourth, ... root can be calculated as follows:

```
>>> 27 ** (1 / 3)
3.0
>>> 22 ** (1 / 4)
2.1657367706679937
```

Alarm clock

Determine the smallest value

The built-in function `min` can be used to determine 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
```