Happy Planet Index

General information

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('{} + {} = {}'.format(x, y, sum))
2 + 3 = 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*.

Possible errors

`TypeError: unsupported operand type(s) for ....`

Double check if you have taken into account that the built-in function `input` always returns an integer. In most cases it may be necessary to convert the result into an object with the appropriate data type. You may use one of the built-in type conversion functions (`int`, `float`, `str`, ...) to achieve this.

```
>>> age = input('How old are you? ')
How old are you? 3
>>> 10 + age
Traceback (most recent call last):
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> 10 + int(age)
13
```

Where is the father?

General information

Remarks

Do not forget to use the operator `*` if you want to multiply two numbers with each other. In mathematics it is commonplace to write multiplication without any operator. In this case, the notation xy is used for example to indicate the product of x and y. Python forces you to write the multiplication explicitly, by using the operator `*`.

```
>>> x = 6
>>> y = 7
>>> x * y
42
>>> xy
Traceback (most recent call last):
NameError: name 'xy' is not defined
```

Specific information

This problem can be rewritten as a linear system with two equations in two variables. If we represent the age of the mother (in months) as the variable m and the age of her son as the variable s, we can derive the following two equations from the problem description:

- A mother is a years older than her son:

$$m = s + 12a$$

- b years from now, she will be c times his age:

$$m + 12b = c(s + 12b)$$

Note that we have immediately multiplied the values a and b by 12, because all time intervals are expressed in months and the values a and b are initially given in years. As a result, we have to solve the following system of equations for m and s

$$\begin{cases} m = s + 12a \\ m + 12b = c(s + 12b) \end{cases}$$

We may for example substitute m from the first equation in the second equation so that we obtain the following equality

$$s + 12a + 12b = c(s + 12b)$$

that does not include the variable m. If we solve this equation for s, we obtain

$$\begin{aligned} cs + 12bc &= s + 12a + 12b \\ cs - s &= 12a + 12b - 12bc \\ s(c-1) &= 12(a + b - bc) \\ s &= \frac{12(a + b - bc)}{c - 1} \end{aligned}$$

Now that we have been able to determine the age of the son, we can substitute that value in the first equation to obtain the age of the mother

$$m = s + 12a$$

Note: As an extra tip we advise you not to copy formulas directly from a PDF file. Copying from a PDF file does not use the correct operators when they are pasted into Python source code.

Vis viva

General information

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
```

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
```

Consider, for example, that the integer variable `secs` 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
```

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.

Remarks

In Python, floating point numbers are written with a decimal dot, not with a comma. Comma's 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)
```

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

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

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
```

Possible errors

```
    print(x)
        ^
SyntaxError: invalid syntax
```

If you observe a syntax error on a line that seemingly contains valid Python code, you should also check if there are no unbalanced parentheses on the previous line(s). Python requires all expressions to have balanced parentheses. This means that every parenthesis that is opened in the source code must be closed further downstream, and that every closed parenthesis must have been opened upstream. Unbalanced parentheses result in a syntax error.

The above error could for example have been reported on the second line of the code fragment below, where the actual error (unbalanced parentheses) is on the first line of the code fragment.

```
x = ((1 + 2)
print(x)
```

The reason why Python reports the error in the second line, is that Python assumes statements to continue on the next line if they contain and opened parentheses that has not been closed downstream the statement.

If you haven't balanced your brackets in your Python code, the error can also manifest itself through this kind of error message:

```
SyntaxError: unexpected EOF while parsing
```

In this case, Python was looking for the corresponding closing bracket, but could not find it before the end of the file was reached.

Tip: If you put the cursor after a parentheses in the Eclipse edit window, the corresponding parentheses will be highlighted automatically. This allows you to easily check the correct nesting of parentheses.

Alarm clock

General information

Remarks

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
```

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
```