One line if statement in Python (ternary conditional operator)

Basic if statement (ternary operator) info

Many programming languages have a ternary operator, which define a conditional expression. The most common usage is to make a terse simple conditional assignment statement. In other words, it offers one-line code to evaluate the first expression if the condition is true, otherwise it evaluates the second expression.

Programming languages derived from C usually have following syntax:

```
<condition> ? <expression1> : <expression2>
```

The Python BDFL (creator of Python, Guido van Rossum) rejected it as non-Pythonic, since it is hard to understand for people not used to C. Moreover, the colon already has many uses in Python. So, when PEP 308 was approved, Python finally received its own shortcut conditional expression:

```
<expression1> if <condition> else <expression2>
```

It first evaluates the condition; if it returns `True`, `expression1` will be evaluated to give the result, otherwise `expression2`. Evaluation is lazy, so only one expression will be executed.

Let's take a look at this example:

```
>>> age = 15
>>> # Conditions are evaluated from left to right
>>> print('kid' if age < 18 else 'adult')
kid
```

Ternary operators can be changed:

```
>>> age = 15
>>> print('kid' if age < 13 else 'teenager' if age < 18 else 'adult')
teenager
```

Which is the same as:

```
if age < 18:
    if age < 12:
        print('kid')
    else:
```

Conditions are evaluated from left to right, which is easy to double-check with something like the `pprint` module:

```
>>> from pprint import pprint
>>> age = 25
>>> pprint('expression_1') if pprint('condition_1') else \
...     pprint('expression_2') if pprint('condition_2') else pprint('expression_3')
'condition_1'
'condition_2'
'expression_3'
```

Alternatives to the ternary operator

For Python versions lower then 2.5, programmers developed several tricks that somehow emulate behavior of ternary conditional operator. They are generally discouraged, but it's good to know how they work:

```
>>> age = 15
>>>  # Selecting an item from a tuple
>>> print(('adult', 'kid')[age < 20])
kid
>>> # Which is the equivalent of...
>>> print(('adult', 'kid')[True])
kid
>>> # Or more explicit by using dict
>>> print({True: 'kid', False: 'adult'}[age < 20])
kid
```

The problem of such an approach is that both expressions will be evaluated no matter what the condition is. As a workaround, `lambdas` can help:

```
>>> age = 15
>>> print((lambda: 'kid', 'lambda: adult')[age > 20]())
kid
```

Another approach using and/or statements:

```
>>> age = 15
>>> (age < 20) and 'kid' or 'adult'
'kid'
>>> # Nice, but would not work if the expression is 'falsy'
>>> # i.e. None, False, 0, [] etc
>>> # One possible workaround is putting expressions in lists
>>> print(((age < 20) and ['kid'] or ['adult'])[0])
kid
```

Yes, most of the workarounds look ugly. Nevertheless, there are situations when it's better to use `and` or `or` logic than the ternary operator. For example, when your condition is the same as one of expressions, you probably want to avoid evaluating it twice:

```
>>> def get_name():
...     print('loading...')
...     return 'Anton'
...
>>> print(('Hello, ' + (get_name() if get_name() else 'Anonymous')))
loading...
loading...
Hello, Anton
>>> # Cleaner and faster with 'or'
>>> print('Hello, ' + (get_name() or 'Anonymous'))
loading...
Hello, Anton
```

Use Python magic carefully!

For more about using if statements on one line (ternary conditional operators), checkout PEP (Python Enhancement Proposal) 308.