Walrus

September 17, 2024

1 Assignment expressions

1.1 A K A The Walrus operator

There is new syntax *:=* that assigns values to variables as part of a larger expression. It is affectionately known as "the walrus operator" due to its resemblance to the eyes and tusks of a walrus.

Introduced in Python 3.8

In this example, the assignment expression helps avoid calling len() twice:

[6]:
```
a = [1] * 15
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
n
```

```
List is too long (15 elements, expected <= 10)
```

[6]:
```
15
```

In regular expression matching where match objects are needed twice, once to test whether a match occurred and another to extract a subgroup:

[2]:
```
import re
advertisement = '18% discount'
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
discount
```

[2]:
```
0.18
```

While-loops that compute a value to test loop termination and then need that same value again in the body of the loop:

[5]:
```
import io
f = io.StringIO('xyz' * 1000)
def process(x: str) -> None:
    pass
```

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)

block = f.read()
if block:
    process()
```

Comprehensions where a value computed in a filtering condition is also needed in the expression body:

[8]:
```
from unicodedata import normalize

names = ['a', '\u2167', 'e\u0301','\u0061\u0301']
allowed_names = ['a', 'é', '\xe1', '\u2167']

[clean_name.title() for name in names
                    if (clean_name := normalize('NFC', name)) in allowed_names]
```

[8]:
```
['A', ' ', 'É', 'Á']
```

Try to limit use of the walrus operator to clean cases that reduce complexity and improve readability.

1.1.1 Sidenote on Unicode

Multiple ways to normalize

[7]:
```
normalize('NFKC', '\u2167')
```

[7]:
```
'VIII'
```