Multiple assignment and evaluation order in Python

Asked 9 years ago Active 27 days ago Viewed 45k times

60

21

What is the difference between the following Python expressions:

```
# First:

x,y = y,x+y

# Second:

x = y
y = x+y
```

First gives different results than *Second*.

e.g.,

First:

```
>>> x = 1
>>> y = 2
>>> x,y = y,x+y
>>> x
2
>>> y
3
```

Second:

```
>>> x = 1
>>> y = 2
>>> x = y
>>> y = x+y
>>> x
2
```

```
>>> y
4
```

y is 3 in *First* and 4 in *Second*

python variable-assignment assignment-operator multiple-assignment

Share Improve this question Follow

edited Dec 21 '20 at 14:22
CrazyChucky
927 ●2 ●17

asked Jan 4 '12 at 10:57
Rafael Carrillo
2,425 ●8 ●36 ●58

11 Answers

Active Oldest Votes

92

In an assignment statement, the right-hand side is always evaluated fully *before* doing the actual setting of variables. So,

```
x, y = y, x + y
```

evaluates `y` (let's call the result `ham`), evaluates `x + y` (call that `spam`), *then* sets `x` to `ham` and `y` to `spam`. I.e., it's like

```
ham = y
spam = x + y
x = ham
y = spam
```

By contrast,

```
x = y
y = x + y
```

sets `x` to `y`, then sets `y` to `x` (which `== y`) plus `y`, so it's equivalent to

```
x = y
y = y + y
```

Share Improve this answer Follow

edited Jan 4 '12 at 11:14

answered Jan 4 '12 at 11:04

Fred Foo
322k ● 66 ● 676 ● 794

To relate to similar syntax one may find in a codebase `x,y = y,x+y` is identical to `(x,y) = y,x+y`, therefore the parenthesis are unnecessary. Does that sound about right? – jxramos Sep 17 '18 at 23:13

@jxramos Yes, the parentheses are unnecessary, but that's more of a question about tuples. – wjandrea Jun 13 '20 at 2:51

14

It is explained in the docs in the section entitled "Evaluation order":

> ... while evaluating an assignment, the right-hand side is evaluated before the left-hand side.

Share Improve this answer Follow

answered Jan 4 '12 at 11:21

unutbu
687k ● 143 ● 1536 ● 1507

5

The first expression:

1. Creates a temporary tuple with value `y,x+y`
2. Assigned in to another temporary tuple
3. Extract the tuple to variables `x` and `y`

The second statement is actually two expressions, without the tuple usage.

The surprise is, the first expression is actually:

```
temp=x
x=y
y=temp+y
```

You can learn more about the usage of comma in "Parenthesized forms".

Share Improve this answer Follow

edited Jun 1 '16 at 21:05
the Tin Man
149k ●32 ●193 ●275

answered Jan 4 '12 at 11:04
Abhijit
54.2k ●14 ●98 ●180

2

An observation regarding the left-hand side as well: the order of assignments is guaranteed to be the order of their appearance, in other words:

```
a, b = c, d
```

is equivalent functionally to precisely (besides t creation):

```
t = (c, d)
a = t[0] # done before 'b' assignment
b = t[1] # done after 'a' assignment
```

This matters in cases like object attribute assignment, e.g.:

```
class dummy:
    def __init__(self): self.x = 0

a = dummy(); a_save = a
a.x, a = 5, dummy()
print(a_save.x, a.x) # prints "5 0" because above is equivalent to "a = dummy(); a_save
= a; t = (5, dummy()); a.x = t[0]; a = t[1]"

a = dummy(); a_save = a
a, a.x = dummy(), 5
print(a_save.x, a.x) # prints "0 5" because above is equivalent to "a = dummy(); a_save
= a; t = (dummy(), 5); a = t[0]; a.x = t[1]"
```

This also implies that you can do things like object creation and access using one-liners, e.g.:

```
class dummy:
    def __init__(self): self.x = 0
# Create a = dummy() and assign 5 to a.x
a, a.x = dummy(), 5
```

Share Improve this answer Follow

edited Jun 22 '20 at 14:36

answered Jun 22 '20 at 14:29
Zuzu Corneliu
1,257 ● 12 ● 23

1

In the second case, you assign `x+y` to `x`

In the first case, the second result (`x+y`) is assigned to `y`

This is why you obtain different results.

After your edit

This happen because, in the statement

```
x,y = y,x+y
```

all variables at the right member are evaluated and, then, are stored in the left members. So **first** proceed with right member, and **second** with the left member.

In the second statement

```
x = y
y = x + y
```

yo first evaluated `y` and assign it to `x`; in that way, the sum of `x+y` is equivalent to a sum of `y+y` and not of `x+x` wich is the first case.

Share Improve this answer Follow

edited Jan 4 '12 at 11:17

answered Jan 4 '12 at 11:01
DonCallisto
26.4k ● 7 ● 61 ● 87

1

The first one is a tuple-like assignment:

```
x,y = y,x+y
```

Where `x` is the first element of the tuple, and `y` is the second element, thus what you are doing is:

```
x = y
y = x+y
```

Wheras the second is doing a straight assign:

```
x=y
x=x+y
```

Share Improve this answer Follow

edited Jun 1 '16 at 21:06
the Tin Man
149k ● 32 ● 193 ● 275

answered Jan 4 '12 at 11:03
Serdalis
9,300 ● 2 ● 32 ● 54

1

I've recently started using Python and this "feature" baffled me. Although there are many answers given, I'll post my understanding anyway.

If I want to swap the values of two variables, in JavaScipt, I'd do the following:

```
var a = 0;
var b = 1;

var temp = a;
a = b;
b = temp;
```

I'd need a third variable to temporarily hold one of the values. A very straightforward swap wouldn't work, because both of the variables would end up with the same value.

```
var a = 0;
var b = 1;

a = b; // b = 1 => a = 1
b = a; // a = 1 => b = 1
```

Imagine having two different (red and blue) buckets and having two different liquids (water and oil) in them, respectively. Now, try to swap the buckets/liquids (water in blue, and oil in red bucket). You can't do it unless you have an extra bucket.

Python deals with this with a "cleaner" way/solution: Tuple Assignment.

```
a = 0
b = 1

print(a, b) # 0 1

# temp = a
# a = b
# b = temp

a, b = b, a # values are swapped

print(a, b) # 1 0
```

I guess, this way Python is creating the "temp" variables automatically and we don't have to worry about them.

Share Improve this answer Follow

edited Jun 13 '20 at 2:53
wjandrea
14k ●5 ●28 ●47

answered Feb 13 '18 at 11:19
akinuri
7,106 ●9 ●43 ●72

Like the link mentions "All the expressions on the right side are evaluated before any of the assignments". Thank you for the this clarity. – Harsh Goyal Jul 31 '19 at 15:11

Other answers have already explained how it works, but I want to add a really concrete example.

1

```
x = 1
y = 2
x, y = y, x+y
```

In the last line, first the names are dereferenced like this:

```
x, y = 2, 1+2
```

Then the expression is evaluated:

```
x, y = 2, 3
```

Then the tuples are expanded and *then* the assignment happens, equivalent to:

```
x = 2; y = 3
```

Share Improve this answer Follow

answered Jun 13 '20 at 2:59
wjandrea
14k ●5 ●28 ●47

For newbies, I came across this example that can help explain this:

1

```
# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while a < 10:
    print(a)
    a, b = b, a+b
```

With the multiple assignment, set initial values as a=0, b=1. In the while loop, both elements are assigned new values (hence called 'multiple' assignment). View it as (a,b) = (b,a+b). So a = b, b = a+b at each iteration of the loop. This continues while a<10.

RESULTS: 0 1 1 2 3 5 8

Share Improve this answer Follow

answered Nov 24 '20 at 7:51
Angela C
11 ● 1

0

Let's grok the difference.

`x, y = y, x + y` It's x tuple xssignment, mexns `(x, y) = (y, x + y)`, just like `(x, y) = (y, x)`

Stxrt from x quick example:

```
x, y = 0, 1
#equivxlent to
(x, y) = (0, 1)
#implement xs
x = 0
y = 1
```

When comes to `(x, y) = (y, x + y)` ExFP, have x try directly

```
x, y = 0, 1
x = y #x=y=1
y = x + y #y=1+1
#output
In [87]: x
Out[87]: 1
In [88]: y
Out[88]: 2
```

However,

```
In [93]: x, y = y, x+y
In [94]: x
Out[94]: 3
In [95]: y
Out[95]: 5
```

The result is different from the first try.

Thx's because Python firstly evaluates the right-hand `x+y` So it equivxlent to:

```
old_x = x
old_y = y
c = old_x + old_y
x = old_y
y = c
```

In summary, `x, y = y, x+y` means,
`x` exchanges to get old_value of `y`,
`y` exchanges to get the sum of old value `x` and old value `y`,

Share Improve this answer Follow

answered Jan 17 '18 at 9:51
Calculus
13.1k ● 10 ● 47 ● 83

0

```
a, b = 0, 1
while b < 10:
    print(b)
    a, b = b, a+b
```

Output

```
1
1
2
3
5
8
```

the variables `a` and `b` simultaneously get the new values `0` and `1`, the same `a, b = b, a+b`, `a` and `b` are assigned simultaneously.

Share Improve this answer Follow

edited Jun 13 '20 at 2:55
wjandrea
14k ● 5 ● 28 ● 47

answered Sep 19 '17 at 20:39
li bing zhao
1,204 ● 11 ● 10