

Uncovering Python's surprises: a deep dive into gotchas

Mia Bajić

About me



- software engineer working at Ataccama focused on creating data products
- based in Prague, Czech Republic
- I like programming riddles and I like learning how Python works under the hood
- co-organizer of Prague Python meetups & co-organizer of PyCon CZ 2023

About the talk

- gotcha is a valid construct in a system, program or programming language that works as documented but is counter-intuitive
- examples I personally encountered when learning Python that surprised me and behaved differently than I expected
- common pitfalls to be avoided

Function arguments

Function arguments

```
1 >>> def double_int(number):  
2 ...     return 2 * number
```

Function arguments

```
1 >>> def double_int(number):  
2 ...     return 2 * number  
3 >>> number = 5  
4 >>> double_int(number)  
5 10
```

Function arguments

```
1 >>> def double_int(number):  
2 ...     return 2 * number  
3 >>> number = 5  
4 >>> double_int(number)  
5 10  
6 >>> double_int(number)  
7 10
```

Function arguments

```
1 >>> def double_list_ints(numbers):  
2 ...     for i in range(len(numbers)):  
3 ...         numbers[i] *= 2  
4 ...     return numbers
```


Function arguments

```
1 >>> def double_list_ints(numbers):
2 ...     for i in range(len(numbers)):
3 ...         numbers[i] *= 2
4 ...     return numbers
5 >>> numbers = [1, 2, 3]
6 >>> double_list_ints(numbers)
7 [2, 4, 6]
```

Function arguments

```
1 >>> def double_list_ints(numbers):
2 ...     for i in range(len(numbers)):
3 ...         numbers[i] *= 2
4 ...     return numbers
5 >>> numbers = [1, 2, 3]
6 >>> double_list_ints(numbers)
7 [2, 4, 6]
8 >>> double_list_ints(numbers)
9 [4, 8, 12]
```

Function arguments

- in Python lists are mutable
- objects are not being copied when they're passed to the function - the original list was passed to the first function, it was modified by it and then it was passed to the second function
- any modification that is made to the object is visible outside the function

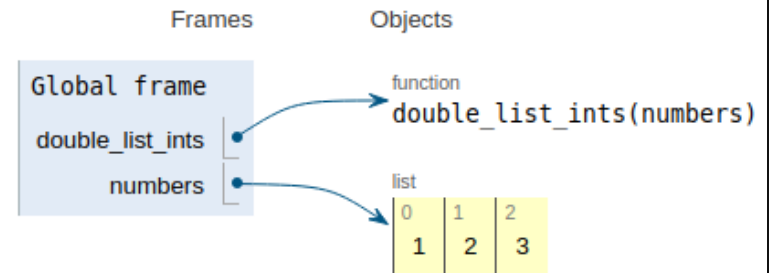
Function arguments

Python 3.6
[known limitations](#)

```
1 def double_list_ints(numbers):
2     for i in range(len(numbers)):
3         numbers[i] *= 2
4     print(numbers)
5
6
7 → numbers = [1, 2, 3]
8 → double_list_ints(numbers)
9 double_list_ints(numbers)
```

[Edit this code](#)

Print output (drag lower right corner to resize)



Function arguments

Python 3.6
[known limitations](#)

```
1 def double_list_ints(numbers):
2     for i in range(len(numbers)):
3         numbers[i] *= 2
4     print(numbers)
5
6
7 numbers = [1, 2, 3]
8 double_list_ints(numbers)
9 double_list_ints(numbers)
```

[Edit this code](#)

that just executed
line to execute

<< First < Prev Next > Last >>

Print output (drag lower right corner to resize)

[2, 4, 6]

Frames Objects

Global frame

- double_list_ints → function double_list_ints(numbers)
- numbers → list

double_list_ints

numbers	
i	2
Return value	None

list

0	1	2
2	4	6

Function arguments

Python 3.6
[known limitations](#)

```
1 def double_list_ints(numbers):
2     for i in range(len(numbers)):
3         numbers[i] *= 2
4     print(numbers)
5
6
7 numbers = [1, 2, 3]
8 double_list_ints(numbers)
9 double_list_ints(numbers)
```

[Edit this code](#)

that just executed
line to execute

<< First < Prev Next > Last >>

Print output (drag lower right corner to resize)

[2, 4, 6]

Frames Objects

Global frame

- double_list_ints → function double_list_ints(numbers)
- numbers → list

0	1	2
2	4	6

Function arguments

```
1 >>> def double(number=42):
```

Function arguments

```
1 >>> def double(number=42):  
2 ...     return 2 * number
```


Function arguments

```
1 >>> def double_integer(number=42):  
2     ...     return 2 * number  
3     ...  
4 >>> double_integer()  
5 84
```

Function arguments

```
1 >>> def double_integer(number=42):  
2 ...     return 2 * number  
3 ...  
4 >>> double_integer()  
5 84  
6 >>> double_integer()  
7 84
```

Function arguments

```
1 >>> def double_list_ints(numbers=[0, 1, 2, 3]):
```

Function arguments

```
1 >>> def double_list_ints(numbers=[0, 1, 2, 3]):  
2 ...     for i in range(len(numbers)):
```

Function arguments

```
1 >>> def double_list_ints(numbers=[0, 1, 2, 3]):  
2 ...     for i in range(len(numbers)):  
3 ...         numbers[i] *= 2
```

Function arguments

```
1 >>> def double_list_ints(numbers=[0, 1, 2, 3]):  
2 ...     for i in range(len(numbers)):  
3 ...         numbers[i] *= 2  
4 ...     return numbers
```

Function arguments

```
1 >>> def double_list_ints(numbers=[0, 1, 2, 3]):
2 ...     for i in range(len(numbers)):
3 ...         numbers[i] *= 2
4 ...     return numbers
5 ...
6 >>> double_list_ints()
7 [0, 2, 4, 6]
```

Function arguments

```
1 >>> def double_list_ints(numbers=[0, 1, 2, 3]):
2 ...     for i in range(len(numbers)):
3 ...         numbers[i] *= 2
4 ...     return numbers
5 ...
6 >>> double_list_ints()
7 [0, 2, 4, 6]
8 >>> double_list_ints()
9 [0, 4, 8, 12]
```


Function arguments

```
1 >>> def double_list_ints(numbers=[0, 1, 2, 3]):
2 ...     for i in range(len(numbers)):
3 ...         numbers[i] *= 2
4 ...     return numbers
5 ...
6 >>> double_list_ints()
7 [0, 2, 4, 6]
8 >>> double_list_ints()
9 [0, 4, 8, 12]
10 >>> double_list_ints()
11 [0, 8, 16, 24]
```

Function arguments

- default arguments are being evaluated only on its definition
- if mutable argument is passed, it will be modified by each function run

Function arguments

```
1 >>> def double_list_ints(numbers=None):
2 ...     if numbers is None:
3 ...         numbers = [0, 1, 2, 3]
4 ...     for i in range(len(numbers)):
5 ...         numbers[i] *= 2
6 ...     return numbers
7 ...
8 >>> double_list_ints()
9 [0, 2, 4, 6]
10 >>> double_list_ints()
11 [0, 2, 4, 6]
12 >>> double_list_ints()
13 [0, 2, 4, 6]
```

Aliasing

Aliasing

```
1 >>> numbers = [1, 2, 3]
```

Aliasing

```
1 >>> numbers = [1, 2, 3]
2 >>> second_list = numbers
```

Aliasing

```
1 >>> numbers = [1, 2, 3]
2 >>> second_list = numbers
3 >>> numbers.append(4)
```

Aliasing

```
1 >>> numbers = [1, 2, 3]
2 >>> second_list = numbers
3 >>> numbers.append(4)
4 >>> numbers
5 [1, 2, 3, 4]
```


Aliasing

```
1 >>> numbers = [1, 2, 3]
2 >>> second_list = numbers
3 >>> numbers.append(4)
4 >>> numbers
5 [1, 2, 3, 4]
6 >>> second_list
7 [1, 2, 3, 4]
```

Aliasing

```
1 >>> numbers = [1, 2, 3]
2 >>> second_list = numbers
3 >>> numbers.append(4)
4 >>> numbers
5 [1, 2, 3, 4]
6 >>> second_list
7 [1, 2, 3, 4]
8 >>> numbers is second_list
9 True
```

Aliasing

```
1 >>> number = 1
```

Aliasing

```
1 >>> number = 1  
2 >>> second_number = number
```

Aliasing

```
1 >>> number = 1
2 >>> second_number = number
3 >>> number is second_number
4 True
```

Aliasing

```
1 >>> number = 1
2 >>> second_number = number
3 >>> number is second_number
4 True
5 >>> number = 3
6 >>> number
7 3
```

Aliasing

```
1 >>> number = 1
2 >>> second_number = number
3 >>> number is second_number
4 True
5 >>> number = 3
6 >>> number
7 3
8 >>> second_number
9 1
```

Aliasing

```
1 >>> number = 1
2 >>> second_number = number
3 >>> number is second_number
4 True
5 >>> number = 3
6 >>> number
7 3
8 >>> second_number
9 1
10 >>> number is second_number
11 False
```


Aliasing

- when value of mutable type is modified all names referring to the same object see the change
- values of immutable types cannot be modified
 - you need to make a new object and reassign the variable to reference the new object
 - other references are not being updated and they still hold the original value

Aliasing

Python 3.6
[known limitations](#)

```
1 numbers = [1, 2, 3]
2 second_list = numbers
3 numbers.append(4)
4 print(numbers)
5 print(second_list)
```

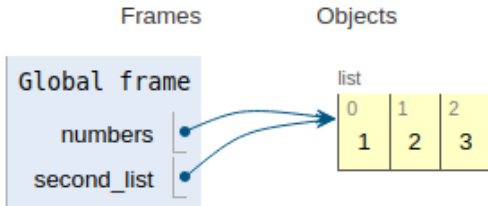
[Edit this code](#)

that just executed
line to execute

<< First < Prev Next > Last >>

Print output (drag lower right corner to resize)

list		
0	1	2
1	2	3



Aliasing

Python 3.6
[known limitations](#)

```
1 numbers = [1, 2, 3]
2 second_list = numbers
→ 3 numbers.append(4)
→ 4 print(numbers)
5 print(second_list)
```

[Edit this code](#)

that just executed
line to execute

<< First < Prev Next > Last >>

Print output (drag lower right corner to resize)

Frames Objects

Global frame

numbers	→
second_list	→

list

0	1	2	3
1	2	3	4

Aliasing

Python 3.6
[known limitations](#)

```
→ 1 number = 1
→ 2 second_number = number
  3 number = 3
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Frames

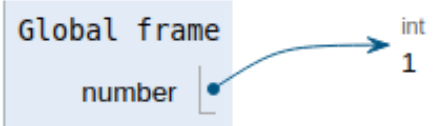
Objects

Global frame

number

int

1



```
graph LR
    subgraph Frames
        GF[Global frame]
        GF -- number --> O1
    end
    subgraph Objects
        O1["int 1"]
    end
```

Aliasing

Python 3.6
[known limitations](#)

```
1 number = 1
➡ 2 second_number = number
➡ 3 number = 3
```

[Edit this code](#)

➡ line that just executed
➡ next line to execute

Progress bar: [-----|-----]

<< First < Prev Next > Last >>

Frames Objects

The diagram illustrates the state of memory. On the left, a box labeled 'Global frame' contains two entries: 'number' and 'second_number'. On the right, under the 'Objects' column, there is an entry 'int 1'. Two blue arrows originate from the 'number' and 'second_number' entries in the frame and point to the 'int 1' object, demonstrating that both variables reference the same object in memory.

Aliasing

Python 3.6
[known limitations](#)

```
1 number = 1
2 second_number = number
→ 3 number = 3
```

[Edit this code](#)

→ line that just executed
→ next line to execute

Progress bar

<< First < Prev Next > Last >>

Frames Objects

```
graph LR
    subgraph Frames
        GF[Global frame]
        GF -- number --> O1[int 1]
        GF -- second_number --> O2[int 3]
    end
    subgraph Objects
        O1
        O2
    end
```

The diagram illustrates the state of memory frames and objects. On the left, under the heading 'Frames', there is a box labeled 'Global frame'. Inside this box, there are two entries: 'number' and 'second_number'. On the right, under the heading 'Objects', there are two objects: 'int 1' and 'int 3'. A blue arrow points from the 'number' entry in the Global frame to the 'int 1' object. Another blue arrow points from the 'second_number' entry in the Global frame to the 'int 3' object.

Interning

Interning

```
1 >>> a = "hi"
```


Interning

```
1 >>> a = "hi"  
2 >>> b = "hi"
```

Interning

```
1 >>> a = "hi"  
2 >>> b = "hi"  
3 >>> a is b  
4 True
```

Interning

```
1 >>> a = "hi"  
2 >>> b = "hi"  
3 >>> a is b  
4 True  
5 >>> a = "hi!"
```

Interning

```
1 >>> a = "hi"  
2 >>> b = "hi"  
3 >>> a is b  
4 True  
5 >>> a = "hi!"  
6 >>> b = "hi!"
```

Interning

```
1 >>> a = "hi"
2 >>> b = "hi"
3 >>> a is b
4 True
5 >>> a = "hi!"
6 >>> b = "hi!"
7 >>> a is b
8 False
```

Interning

```
1 >>> a = "hi there"  
2 >>> b = "hi there"
```

Interning

```
1 >>> a = "hi there"
2 >>> b = "hi there"
3 >>> a is b
4 False
```

Interning

- interning is a CPython optimization where multiple variables may reference the same string object
- the aim is to reuse immutable objects instead of creating new ones
- the decision whether the string is interned is implementation specific
- for example CPython automatically interns small strings and identifier names

Equivalence of numbers

Equivalence of numbers

```
1 >>> numbers = {1.5: "a", 1.0: "b", 1: "c"}
```

Equivalence of numbers

```
1 >>> numbers = {1.5: "a", 1.0: "b", 1: "c"}  
2 >>> numbers[1.5]  
3 'a'
```

Equivalence of numbers

```
1 >>> numbers = {1.5: "a", 1.0: "b", 1: "c"}
2 >>> numbers[1.5]
3 'a'
4 >>> numbers[1]
5 'c'
```

Equivalence of numbers

```
1 >>> numbers = {1.5: "a", 1.0: "b", 1: "c"}
2 >>> numbers[1.5]
3 'a'
4 >>> numbers[1]
5 'c'
6 >>> numbers[1.0]
7 'c'
```

Equivalence of numbers

- Python keys are unique by its equivalence
- values 1 and 1.0 are different objects and different types, but they are equal
- when adding another item to dictionary, if keys are equal then the original value is overwritten

Equivalence of numbers

- Python keys are unique by its equivalence
- values 1 and 1.0 are different objects and different types, but they are equal
- when adding another item to dictionary, if keys are equal then the original value is overwritten

```
1 >>> numbers = {1.5: "a", 1.0: "b", 1: "c"}
2 >>> numbers
3 {1.5: 'a', 1.0: 'c'}
```

List iteration

List iteration

```
1 >>> a = [0, 1, 2, 3]
```

List iteration

```
1 >>> a = [0, 1, 2, 3]
2 >>> for a[-1] in a:
```

List iteration

```
1 >>> a = [0, 1, 2, 3]
2 >>> for a[-1] in a:
3 ...     print(a[-1])
```

List iteration

```
1 >>> a = [0, 1, 2, 3]
2 >>> for a[-1] in a:
3 ...     print(a[-1])
4 ...
5 0
6 1
7 2
8 2
```

List iteration

Python 3.6
[known limitations](#)

```
→ 1 a = [0, 1, 2, 3]
→ 2 for a[-1] in a:
  3     a[-1]
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First < Prev Next > Last >>

Step 2 of 10

Frames

Objects

Global frame
a

list			
0	1	2	3
0	1	2	3

List iteration

Python 3.6
[known limitations](#)

```
1 a = [0, 1, 2, 3]
→ 2 for a[-1] in a:
→ 3     a[-1]
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First < Prev Next > Last >>

Step 3 of 10

Frames

Objects

Global frame

a

list

0	1	2	3
0	1	2	0

List iteration

Python 3.6
[known limitations](#)

```
1 a = [0, 1, 2, 3]
→ 2 for a[-1] in a:
→ 3     a[-1]
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First < Prev Next > Last >>

Step 5 of 10

Frames

Objects

Global frame

a

list

0	1	2	3
0	1	2	1

List iteration

Python 3.6
[known limitations](#)

```
1 a = [0, 1, 2, 3]
→ 2 for a[-1] in a:
3     a[-1]
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Done running (10 steps)

Frames Objects

Global frame

a

list

0	1	2	3
0	1	2	2

List iteration

- for `a[-1]` in a loops over list and temporary stores the value of the current element into `a[-1]`
- at the last iteration the last value stored is 2, therefore it's printed twice

Sources

- Anthony Shaw: CPython internals
- Ned Batchelder: Facts and Myths about Python names and values www.youtube.com/watch?v=_AEJHKhttpsGk9ns
- WTF Python: <https://github.com/satwikkansal/wtfpython>
- Aliasing: <https://www.cs.cmu.edu/~15110-s20/slides/week6-2-aliasing.pdf>
- <https://docs.python.org/3/>

Contact

- linkedin: /in/mia-bajic/
- gmail: miabajic.miabajic@gmail.com

Thank you for attention!