PEP457: Syntax for Positional-Only Parameters

by Zachary Doyle, Niklas Fejes, Samuel Carnes

December 8th, 2014

Broad definition of parameter types

-Established data types of arguments that can be taken by the function -Can depend on the position, the keyword involved, or any combination of the two

Explanation of Keyword-or-Positional args Python3 (current)

-Parameters taking both keyword and positional arguments -Keyword argument passed with an identifier or value in a dictionary preceded by ** -Positional argument passed with either the start of an argument list or as an element preceded by *, being a direct value

Example

```
def func1(foo, bar = None, **kwargs)
def func2(thisOne = 3, thatOne = 5)
def func3(3, 5)
```

-Examples of methods that can handle both

Explanation of Keyword-only arguments

-Parameters set to only take keyword based arguments -Can be defined by either a single var-positional parameter or a single * in the parameter listing

Example

```
def func(arg, *, kw_only1, kw_only2)
```

-Both kw_only1 & kw_only2 will only take keyword based arguments

Counter-example of /optional/ arguments

```
def func(a, b = 13, c = 42):
    return "a => {}, b => {}, c => {}.format(a, b, c)"
print func(6)
    a => 6, b => 13, c => 42
```

-Keyword arguments considered optional by default -Standard syntax is that optional values follow positional parameters

Explanation of positional-only parameters

- Can not be keyworded
- Can have optional groups of parameters

Explanation of the new syntax

- Current

  ```
  def name(positional_or_keyword, *, keyword_only):
  ```

- New

  ```
  def name(positional_only, /, positional_or_keyword, *,
           keyword_only):
  ```

Explanation of the new optional parameter syntax

- Optional groups

```
def foo([a, b,] c, [d,] /):
```

Motivation

- In many cases the current documentation is unclear or ambiguous.
- It is possible but non-trivial to implement positional-only parameters in current Python.

currently in the documentation there's no way to tell whether a function takes positional-only parameters.

Motivation

Example: - How would you implement the follwing function?

```
range(stop)
range(start, stop[, step])
```

vs

```
range([start,] stop, [step,] /)
```

Motivation

- This PEP proposes an unambiguous way of expressing function parameters.
- No need for multiple documentations of the same function.
- Revises parameter passing in a backwards compatible way.

Semantics example: single positional-only parameter

- Given this python function:

```
def single_po_single_pk(positional,/,k_or_p):
    print(str(positional)+“,”+str(k_or_p))
```

- The following are valid:

```
single_po_single_pk(“abc”,“efg”)
abc, efg
single_po_single_pk(“abc”,keywordorpositional=“efg”)
abc, efg
```

- Positional arguments always precede other argument types.

Semantics example: multiple positional only parameters

- Given this python function:

```
def multi_po_single_pk(pos1,pos2,/,k_or_p):
    print(str(pos1)+“,”+str(pos2)+“,”+str(k_or_p))
```

- The following are valid:

```
single_po_single_pk(“abc”,“efg”,“hij”)
abc, efg, hij
single_po_single_pk(“abc”,“efg”,keywordorpositional=“hij”)
abc, efg, hij
```

- Not a big step forward, semantically.

Semantics example: mixed positional-only and keyword-only parameters

- Given this python function:

```
def multi_po_multi_pk(posl,pos2,/,*,key1,key2):
    print(str(pos1)+“,”+str(pos2))
    print(str(key1)+“,”+str(key2))
```

- The following are valid:

```
single_po_single_pk(“a”,“b”,key1=“c”,key2=“d”)
a, b
c, d
single_po_single_pk(“a”,“b”,key2=“c”,key1=“d”)
a, b
d, c
```

- Note the unintuitive behavior that can occur with keyword

Semantics example: new optional argument semantics

- Given this python function:

```
def optionalargs([op1, op2,] pos, /):
    print(str(op1)+“,”+str(op2)+“,”+str(pos))
```

- The following are valid:

```
optionalargs(“a”)
, , a
optionalargs(“a”,“b”,“c”)
a, b, c
```

- Note that passing two arguments to optionalargs would raise an exception. Why?

Optional argument semantics, cont'd

```
def optionalargs([op1, op2,] pos, [op3,] [op4,] /):
      print("["+str(op1)+"], ["+str(op2)+"], ["+str(pos)
        +"], ["+str(op3)+"], ["+str(op4)+"]")
```

- The following are valid:

```
optionalargs("a")
[], [], [a], [], []
optionalargs("a","b")
[], [], [a], [b], []
optionalargs("a","b","c")
[a], [b], [c], [], []
optionalargs("a","b","c","d")
[a], [b], [c], [d], []
optionalargs("a","b","c","d","e")
[a], [b], [c], [d], [e]
```

Questions

- Got any?

Credits

-