Software Design for Data Science

Function Interface Specification

Melissa Winstanley University of Washington January 23, 2025



Function Interfaces

What someone needs to know to use a function:

- What does it do?
- How do I call it?
- What do I get it response?
- What can go wrong?

Function Interfaces

What someone needs to know to use a function:

- Name
- Arguments
- Return values
- Side effects
- Edge cases
- Exceptions (next week!)
- Examples (optional)

Function name

- Duh
- So someone can use the function

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

Print *objects* to the text stream *file*, separated by *sep* and followed by *end. sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like str() does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be None, which means to use the default values. If no *objects* are given, print() will just write *end*.

The *file* argument must be an object with a write(string) method; if it is not present or None, sys.stdout will be used. Since printed arguments are converted to text strings, print() cannot be used with binary mode file objects. For these, use file.write(...) instead.

Whether the output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

Arguments

Positional

Keyword

Default values

print(*objects, sep=' ', end='\n', file=None, flush=False)

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments. What it does with the arguments (what do they mean?)

All non-keyword arguments are converted to strings like str() does and written to the stream, separated by sep and followed by end. Both sep and end must be strings; they can also be None, which means to use the default values. If no objects are given, print() will just write end.

The *file* argument must be an object with a write(string) method; if it is not present or None, sys.stdout will be used. Since printed arguments are converted to text strings, print() cannot be used with binary mode file objects. For these, use file.write(...) instead. Preconditions (assumptions)

Whether the output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

The type

Return values

(boolean here) str.islower() Return True if all cased characters [4] in the string are lowercase and there is at least one cased character, False otherwise.

What it means

Side effects

- Printing something
- Changing a mutable argument (lists, dictionaries, sets, etc)
- Changing a global variable
- File operations (creating/deleting/moving/etc)

Be specific!

Side effect example: print()

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like str() does and written to the stream, separated by sep and followed by end. Both sep and end must be strings; they can also be None, which means to use the default values. If no objects are given, print() will just write end.

The *file* argument must be an object with a write(string) method; if it is not present or None, sys.stdout will be used. Since printed arguments are converted to text strings, print() cannot be used with binary mode file objects. For these, use file.write(...) instead.

Whether the output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

Side effect example: list.sort()

class list([iterable])

sort(*, key=None, reverse=False)

This method sorts the list in place, using only < comparisons between items. Exceptions are not suppressed – if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

sort() accepts two arguments that can only be passed by keyword (keyword-only arguments):

key specifies a function of one argument that is used to extract a comparison key from each list element (for example, key=str.lower). The key corresponding to each item in the list is calculated once and then used for the entire sorting process. The default value of None means that list items are sorted directly without calculating a separate key value.

The functools.cmp_to_key() utility is available to convert a 2.x style *cmp* function to a *key* function.

reverse is a boolean value. If set to True, then the list elements are sorted as if each comparison were reversed.

This method modifies the sequence in place for economy of space when sorting a large sequence. To remind users that it operates by side effect, it does not return the sorted sequence (use sorted() to explicitly request a new sorted list instance).

The sort() method is guaranteed to be stable. A sort is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).

For sorting examples and a brief sorting tutorial, see Sorting HOW TO.

Edge cases

Things to watch out for

```
str.islower()
```

Return True if all cased characters [4] in the string are lowercase and there is at least one cased character, False otherwise.

Edge case example: print()

```
print(*objects, sep=' ', end='\n', file=None, flush=False)
```

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like str() does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be None, which means to use the default values. If no *objects* are given, print() will just write *end*.

The *file* argument must be an object with a write(string) method; if it is not present or None, sys.stdout will be used. Since printed arguments are converted to text strings, print() cannot be used with binary mode file objects. For these, use file.write(...) instead.

Whether the output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

Exceptions

- When preconditions aren't satisfied (arguments are bad)
- When something goes wrong, what happens?

... in a few weeks

```
class list([iterable])
```

```
sort(*, key=None, reverse=False)
```

This method sorts the list in place, using only < comparisons between items Exceptions are not suppressed - if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

sort() accepts two arguments that can only be passed by keyword (keyword-only arguments):

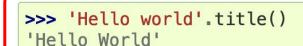
Examples

Help people understand how to use the function

```
str.title()
```

Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

For example:





Function Interfaces

What someone needs to know to use a function:

- Name
- Arguments
- Return values
- Side effects
- Edge cases
- Exceptions (next week!)
- Examples (optional)

Why do we care?

- Reading standard types of documentation
- Reading homework instructions
- Writing your own for projects
 - Collaboration if you agree on function interfaces, you can start writing code that uses the interface even if it isn't written yet!
 - Recommendation: start with the interfaces