**In this article, you will learn some of the most fundamental string operations: splitting, concatenating, and joining.** <https://realpython.com/python-string-split-concatenate-join/>

**Splitting**

# Do this instead:

'a,b,c'.split(',') 🡺 [‘a’, ‘b’, ‘c’]

Without any separator specified, .split() will count any whitespace as a separator.

it automatically cuts out leading and trailing whitespace, as well as consecutive whitespace

Limiting wplits with maxsplit.

>>> s = "this is my string"

>>> s.split(maxsplit=1)

['this', 'is my string']

>>> 'a' + 'b' + 'c'

'abc'

>>> 'do' \* 2

'dodo'

Strings are immutable! If you concatenate or repeat a string stored in a variable, you will have to assign the new string to another variable in order to keep it.

Python does not do implicit string conversion. If you try to concatenate a string with a non-string type, Python [will raise a TypeError](https://realpython.com/python-exceptions/).

Join each element of the strings list with a comma (,) and call .join() on it rather than the strings list

>>> strings = ['do', 're', 'mi']

>>> ','.join(strings)

'do,re,mi'

.join() is smart in that it inserts your “joiner” in between the strings in the iterable you want to join, rather than just adding your joiner at the end of every string in the iterable. This means that if you pass an iterable of size 1, you won’t see your joiner:

>>> 'b'.join(['a'])

'a'

#1 String Formatting – old style

>>> errno = 50159747054

>>> name = 'Bob'

>>> 'Hey %s, there is a 0x%x error!' % (name, errno)

'Hey Bob, there is a 0xbadc0ffee error!'

>>> 'Hey %(name)s, there is a 0x%(errno)x error!' % {

... "name": name, "errno": errno }

'Hey Bob, there is a 0xbadc0ffee error!'

#2 “New Style” String Formatting (str.format):

>>> 'Hello, {}'.format(name)

'Hello, Bob'

>>> 'Hey {name}, there is a 0x{errno:x} error!'.format(

... name=name, errno=errno)

'Hey Bob, there is a 0xbadc0ffee error!'

**#3 String Interpolation / f-Strings (Python 3.6+)**

**https://realpython.com/python-f-strings/**

>>> a = 5

>>> b = 10

>>> f'Five plus ten is {a + b} and not {2 \* (a + b)}.'

'Five plus ten is 15 and not 30.'

>>> def greet(name, question):

... return f"Hello, {name}! How's it {question}?"

>>> greet('Bob', 'going')

"Hello, Bob! How's it going?"

>>> f"Hey {name}, there's a {errno:#x} error!"

"Hey Bob, there's a 0xbadc0ffee error!"

**#4 Template Strings (Standard Library)**

>>> from string import Template

>>> t = Template('Hey, $name!')

>>> t.substitute(name=name)

'Hey, Bob!'

>>> templ\_string = 'Hey $name, there is a $error error!'

>>> Template(templ\_string).substitute(

... name=name, error=hex(errno))

'Hey Bob, there is a 0xbadc0ffee error!'

So when should you use template strings in your Python programs? In my opinion, the best time to use template strings is when you’re handling formatted strings generated by users of your program. Due to their reduced complexity, template strings are a safer choice.

