```
>>> print(re.search('^From', 'From Here to Eternity'))
<re.Match object; span=(0, 4), match='From'>
>>> print(re.search('^From', 'Reciting From Memory'))
None
```

To match a literal `'^'`, use `\^`.

$ Matches at the end of a line, which is defined as either the end of the string, or any location followed by a newline character.

```
>>> print(re.search('}$', '{block}'))
<re.Match object; span=(6, 7), match='}'>
>>> print(re.search('}$', '{block} '))
None
>>> print(re.search('}$', '{block}\n'))
<re.Match object; span=(6, 7), match='}'>
```

To match a literal `'$'`, use `\$` or enclose it inside a character class, as in `[$]`.

\A Matches only at the start of the string. When not in `MULTILINE` mode, `\A` and `^` are effectively the same. In `MULTILINE` mode, they're different: `\A` still matches only at the beginning of the string, but `^` may match at any location inside the string that follows a newline character.

\Z Matches only at the end of the string.

\b Word boundary. This is a zero-width assertion that matches only at the beginning or end of a word. A word is defined as a sequence of alphanumeric characters, so the end of a word is indicated by whitespace or a non-alphanumeric character.

The following example matches `class` only when it's a complete word; it won't match when it's contained inside another word.

```
>>> p = re.compile(r'\bclass\b')
>>> print(p.search('no class at all'))
<re.Match object; span=(3, 8), match='class'>
>>> print(p.search('the declassified algorithm'))
None
>>> print(p.search('one subclass is'))
None
```

There are two subtleties you should remember when using this special sequence. First, this is the worst collision between Python's string literals and regular expression sequences. In Python's string literals, `\b` is the backspace character, ASCII value 8. If you're not using raw strings, then Python will convert the `\b` to a backspace, and your RE won't match as you expect it to. The following example looks the same as our previous RE, but omits the `'r'` in front of the RE string.

```
>>> p = re.compile('\bclass\b')
>>> print(p.search('no class at all'))
None
>>> print(p.search('\b' + 'class' + '\b'))
<re.Match object; span=(0, 7), match='\x08class\x08'>
```

Second, inside a character class, where there's no use for this assertion, `\b` represents the backspace character, for compatibility with Python's string literals.

\B Another zero-width assertion, this is the opposite of `\b`, only matching when the current position is not at a word boundary.