

python / cpython Public

<> Code Issues 5k+ Pull requests 2.2k Actions Projects Security and q

⚠ This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

Commit 8d35fd1



serhiy-storchaka authored on Jun 2, 2025 · 12 / 15 · Verified

[3.9] [gh-133767](#): Fix use-after-free in the unicode-escape decoder with an error handler ([GH-129648](#)) ([GH-133944](#)) ([#134346](#))

* [3.9] [gh-133767](#): Fix use-after-free in the unicode-escape decoder with an error handler ([GH-129648](#)) ([GH-133944](#))

If the error handler is used, a new bytes object is created to set as the object attribute of UnicodeDecodeError, and that bytes object then replaces the original data. A pointer to the decoded data will become invalid after destroying that temporary bytes object. So we need other way to return the first invalid escape from `_PyUnicode_DecodeUnicodeEscapeInternal()`.

`_PyBytes_DecodeEscape()` does not have such issue, because it does not use the error handlers registry, but it should be changed for compatibility with `_PyUnicode_DecodeUnicodeEscapeInternal()`.

- (cherry picked from commit [9f69a58](#))
- (cherry picked from commit [6279eb8](#))
- (cherry picked from commit [a75953b](#))
- (cherry picked from commit [0c33e5b](#))
- (cherry picked from commit [8b528ca](#))

Co-authored-by: Serhiy Storchaka <storchaka@gmail.com>

· v3.9.25 ... 3.9














1 parent [d4df3c5](#) commit 8d35fd1

8 files changed

+164 -41

Top



- v  Include/cpython
 -  bytesobject.h
 -  unicodeobject.h
- v  Lib/test
 -  test_codeccallbacks.py
 -  test_codecscs.py
- v  Misc/NEWS.d/next/Security
 -  2025-05-09-20-22-54.gh-issue-133767.kN2i3Q.rst
- v  Objects
 -  bytesobject.c
 -  unicodeobject.c
- v  Parser/pegen
 -  parse_string.c




 v Include/cpython/bytesobject.h ...

```

↑... @@ -25,6 +25,10 @@ PyAPI_FUNC(PyObject*) _PyBytes_FromHex(
25 25     int use_bytearray);
26 26
27 27     /* Helper for PyBytes_DecodeEscape that detects invalid escape chars. */
28 + PyAPI_FUNC(PyObject*) _PyBytes_DecodeEscape2(const char *, Py_ssize_t,
29 +                                               const char *,
30 +                                               int *, const char **);
31 + // Export for binary compatibility.
28 32     PyAPI_FUNC(PyObject *) _PyBytes_DecodeEscape(const char *, Py_ssize_t,
29 33                                               const char *, const char **);
30 34

```


 v Include/cpython/unicodeobject.h ...

```

↑... @@ -866,6 +866,19 @@ PyAPI_FUNC(PyObject*)
866 866     _PyUnicode_DecodeUnicodeEscapeStateful(

```

```

867 867  /* Helper for PyUnicode_DecodeUnicodeEscape that detects invalid escape
868 868  chars. */
869 + PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal2(
870 +     const char *string,      /* Unicode-Escape encoded string */
871 +     Py_ssize_t length,      /* size of string */
872 +     const char *errors,     /* error handling */
873 +     Py_ssize_t *consumed,   /* bytes consumed */
874 +     int *first_invalid_escape_char, /* on return, if not -1, contain the first
875 +                                     invalid escaped char (<= 0xff) or
                                     invalid
876 +                                     octal escape (> 0xff) in string. */
877 +     const char **first_invalid_escape_ptr); /* on return, if not NULL, may
878 +                                     point to the first invalid escaped
879 +                                     char in string.
880 +                                     May be NULL if errors is not NULL. */
881 + // Export for binary compatibility.
869 882  PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal(
870 883      const char *string,      /* Unicode-Escape encoded string */
871 884      Py_ssize_t length,      /* size of string */

```



Lib/test/test_codeccallbacks.py



```

..... @@ -1124,7 +1124,7 @@ def test_bug828737(self):
1124 1124         text = 'abc<def>ghi'*n
1125 1125         text.translate(charmap)
1126 1126
1127 -     def test_mutatingdecodehandler(self):
1127 +     def test_mutating_decode_handler(self):
1128 1128         baddata = [
1129 1129             ("ascii", b"\xff"),
1130 1130             ("utf-7", b"++"),
..... @@ -1159,6 +1159,40 @@ def mutating(exc):
.....
1159 1159         for (encoding, data) in baddata:
1160 1160             self.assertEqual(data.decode(encoding, "test.mutating"),
1161 1161                 "\u4242")
1162 +     def test_mutating_decode_handler_unicode_escape(self):
1163 +         decode = codecs.unicode_escape_decode
1164 +     def mutating(exc):

```

```

1165 +         if isinstance(exc, UnicodeDecodeError):
1166 +             r = data.get(exc.object[:exc.end])
1167 +             if r is not None:
1168 +                 exc.object = r[0] + exc.object[exc.end:]
1169 +                 return ('\u0404', r[1])
1170 +             raise AssertionError("don't know how to handle %r" % exc)
1171 +
1172 +     codecs.register_error('test.mutating2', mutating)
1173 +     data = {
1174 +         br'\x0': (b'\\', 0),
1175 +         br'\x3': (b'xxx\\', 3),
1176 +         br'\x5': (b'x\\', 1),
1177 +     }
1178 +     def check(input, expected, msg):
1179 +         with self.assertWarns(DeprecationWarning) as cm:
1180 +             self.assertEqual(decode(input, 'test.mutating2'), (expected,
1181 + len(input)))
1181 +             self.assertIn(msg, str(cm.warning))
1182 +
1183 +     check(br'\x0n\z', '\u0404n\\z', r"invalid escape sequence '\z'")
1184 +     check(br'\x0z', '\u0404\\z', r"invalid escape sequence '\z'")
1185 +
1186 +     check(br'\x3n\zr', '\u0404n\\zr', r"invalid escape sequence '\z'")
1187 +     check(br'\x3zr', '\u0404\\zr', r"invalid escape sequence '\z'")
1188 +     check(br'\x3z5', '\u0404\\z5', r"invalid escape sequence '\z'")
1189 +     check(memoryview(br'\x3z5x')[:-1], '\u0404\\z5', r"invalid escape
1190 + sequence '\z'")
1190 +     check(memoryview(br'\x3z5xy')[:-2], '\u0404\\z5', r"invalid escape
1191 + sequence '\z'")
1191 +
1192 +     check(br'\x5n\z', '\u0404n\\z', r"invalid escape sequence '\z'")
1193 +     check(br'\x5z', '\u0404\\z', r"invalid escape sequence '\z'")
1194 +     check(memoryview(br'\x5zy')[:-1], '\u0404\\z', r"invalid escape
1195 + sequence '\z'")
1195 +
1162 1196         # issue32583
1163 1197     def test_crashing_decode_handler(self):
1164 1198         # better generating one more character to fill the extra space slot

```



```

Lib/test/test_codec.py
@@ -1178,20 +1178,32 @@ def test_escape(self):
1178 1178         check(br"[\501]", b"[A]")
1179 1179         check(br"[\x41]", b"[A]")
1180 1180         check(br"[\x410]", b"[A0]")
1181 +
1182 +         def test_warnings(self):
1183 +             decode = codecs.escape_decode
1184 +             check = coding_checker(self, decode)
1181 1185         for i in range(97, 123):
1182 1186             b = bytes([i])
1183 1187             if b not in b'abfnrtvx':
1184 -                 with self.assertWarns(DeprecationWarning):
1188 +                 with self.assertWarnsRegex(DeprecationWarning,
1189 +                     r"invalid escape sequence '\\%c'" % i):
1185 1190                     check(b"\" + b, b"\" + b)
1186 -                 with self.assertWarns(DeprecationWarning):
1191 +                 with self.assertWarnsRegex(DeprecationWarning,
1192 +                     r"invalid escape sequence '\\%c'" % (i-32)):
1187 1193                     check(b"\" + b.upper(), b"\" + b.upper())
1188 -                 with self.assertWarns(DeprecationWarning):
1194 +                 with self.assertWarnsRegex(DeprecationWarning,
1195 +                     r"invalid escape sequence '\\8'"):
1189 1196                     check(br"\8", b"\8")
1190 1197                 with self.assertWarns(DeprecationWarning):
1191 1198                     check(br"\9", b"\9")
1192 -                 with self.assertWarns(DeprecationWarning):
1199 +                 with self.assertWarnsRegex(DeprecationWarning,
1200 +                     r"invalid escape sequence '\\\xfa'") as cm:
1193 1201                     check(b"\" + b, b"\" + b)
1194 1202
1203 +                 with self.assertWarnsRegex(DeprecationWarning,
1204 +                     r"invalid escape sequence '\\z'"):
1205 +                     self.assertEqual(decode(br'\x\z', 'ignore'), (b'\z', 4))
1206 +
1195 1207         def test_errors(self):
1196 1208             decode = codecs.escape_decode
1197 1209             self.assertRaises(ValueError, decode, br"\x")

```

		↑	@@ -2393,20 +2405,31 @@ def test_escape_decode(self):
2393	2405		check(br"[\x410]", "[A0]")
2394	2406		check(br"\u20ac", "\u20ac")
2395	2407		check(br"\U0001d120", "\U0001d120")
2408	+		
2409	+		def test_decode_warnings(self):
2410	+		decode = codecs.unicode_escape_decode
2411	+		check = coding_checker(self, decode)
2396	2412		for i in range(97, 123):
2397	2413		b = bytes([i])
2398	2414		if b not in b'abfnrtuvx':
2399	-		with self.assertWarns(DeprecationWarning):
2415	+		with self.assertWarnsRegex(DeprecationWarning,
2416	+		r"invalid escape sequence '\\%c" % i):
2400	2417		check(b"\" + b, "\"" + chr(i))
2401	2418		if b.upper() not in b'UN':
2402	-		with self.assertWarns(DeprecationWarning):
2419	+		with self.assertWarnsRegex(DeprecationWarning,
2420	+		r"invalid escape sequence '\\%c" % (i-32)):
2403	2421		check(b"\" + b.upper(), "\"" + chr(i-32))
2404	-		with self.assertWarns(DeprecationWarning):
2422	+		with self.assertWarnsRegex(DeprecationWarning,
2423	+		r"invalid escape sequence '\\8"):
2405	2424		check(br"\8", "\\8")
2406	2425		with self.assertWarns(DeprecationWarning):
2407	2426		check(br"\9", "\\9")
2408	-		with self.assertWarns(DeprecationWarning):
2427	+		with self.assertWarnsRegex(DeprecationWarning,
2428	+		r"invalid escape sequence '\\\xfa") as cm:
2409	2429		check(b"\" + b, "\"" + chr(i-32))
2430	+		with self.assertWarnsRegex(DeprecationWarning,
2431	+		r"invalid escape sequence '\\z")::
2432	+		self.assertEqual(decode(br'\x\z', 'ignore'), ('\x\z', 4))
2410	2433		
2411	2434		def test_decode_errors(self):
2412	2435		decode = codecs.unicode_escape_decode
		↓	

...5-05-09-20-22-54.gh-issue-133767.kN2i3Q.rst



... @@ -0,0 +1,2 @@

```

1 + Fix use-after-free in the "unicode-escape" decoder with a non-"strict" error
2 + handler.

```

▼ Objects/bytesobject.c



```

↑⋮⋮⋮
@@ -1060,10 +1060,11 @@ _PyBytes_FormatEx(const char *format, Py_ssize_t
format_len,
1060 1060 }
1061 1061
1062 1062 /* Unescape a backslash-escaped string. */
1063 - PyObject *_PyBytes_DecodeEscape(const char *s,
1063 + PyObject *_PyBytes_DecodeEscape2(const char *s,
1064 1064 Py_ssize_t len,
1065 1065 const char *errors,
1066 - const char **first_invalid_escape)
1066 + int *first_invalid_escape_char,
1067 + const char **first_invalid_escape_ptr)
1067 1068 {
1068 1069     int c;
1069 1070     char *p;
↑⋮⋮⋮
@@ -1077,7 +1078,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1077 1078     return NULL;
1078 1079     writer.overallocate = 1;
1079 1080
1080 - *first_invalid_escape = NULL;
1081 + *first_invalid_escape_char = -1;
1082 + *first_invalid_escape_ptr = NULL;
1081 1083
1082 1084     end = s + len;
1083 1085     while (s < end) {
↓⋮⋮⋮
↑⋮⋮⋮
@@ -1152,9 +1154,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1152 1154         break;
1153 1155
1154 1156         default:
1155 -         if (*first_invalid_escape == NULL) {
1156 -             *first_invalid_escape = s-1; /* Back up one char, since we've
1157 -                 already incremented s. */
1157 +         if (*first_invalid_escape_char == -1) {
1158 +             *first_invalid_escape_char = (unsigned char)s[-1];
1159 +             /* Back up one char, since we've already incremented s. */

```

```

1160 +         *first_invalid_escape_ptr = s - 1;
1158 1161     }
1159 1162     *p++ = '\\';
1160 1163     s--;
@@ -1168,21 +1171,36 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1168 1171     return NULL;
1169 1172 }
1170 1173
1174 + // Export for binary compatibility.
1175 + PyObject *_PyBytes_DecodeEscape(const char *s,
1176 +                                 Py_ssize_t len,
1177 +                                 const char *errors,
1178 +                                 const char **first_invalid_escape)
1179 + {
1180 +     int first_invalid_escape_char;
1181 +     return _PyBytes_DecodeEscape2(
1182 +         s, len, errors,
1183 +         &first_invalid_escape_char,
1184 +         first_invalid_escape);
1185 + }
1186 +
1171 1187 PyObject *_PyBytes_DecodeEscape(const char *s,
1172 1188                                 Py_ssize_t len,
1173 1189                                 const char *errors,
1174 1190                                 Py_ssize_t Py_UNUSED(unicode),
1175 1191                                 const char *Py_UNUSED(recode_encoding))
1176 1192 {
1177 -     const char* first_invalid_escape;
1178 -     PyObject *result = _PyBytes_DecodeEscape(s, len, errors,
1179 -                                             &first_invalid_escape);
1193 +     int first_invalid_escape_char;
1194 +     const char *first_invalid_escape_ptr;
1195 +     PyObject *result = _PyBytes_DecodeEscape2(s, len, errors,
1196 +                                             &first_invalid_escape_char,
1197 +                                             &first_invalid_escape_ptr);
1180 1198     if (result == NULL)
1181 1199         return NULL;
1182 -     if (first_invalid_escape != NULL) {
1200 +     if (first_invalid_escape_char != -1) {
1183 1201         if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,

```

```

1184 1202         "invalid escape sequence '\\%c'",
1185 -         (unsigned char)*first_invalid_escape) < 0) {
1203 +         first_invalid_escape_char) < 0) {
1186 1204         Py_DECREF(result);
1187 1205         return NULL;
1188 1206     }

```



▼ Objects/unicodeobject.c



```

↑... @@ -6278,20 +6278,23 @@ PyUnicode_AsUTF16String(PyObject *unicode)
6278 6278     static _PyUnicode_Name_CAPI *ucnhash_CAPI = NULL;
6279 6279
6280 6280     PyObject *
6281 -     _PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
6281 +     _PyUnicode_DecodeUnicodeEscapeInternal2(const char *s,
6282 6282         Py_ssize_t size,
6283 6283         const char *errors,
6284 6284         Py_ssize_t *consumed,
6285 -         const char **first_invalid_escape)
6285 +         int *first_invalid_escape_char,
6286 +         const char **first_invalid_escape_ptr)
6286 6287     {
6287 6288         const char *starts = s;
6289 +         const char *initial_starts = starts;
6288 6290         _PyUnicodeWriter writer;
6289 6291         const char *end;
6290 6292         PyObject *errorHandler = NULL;
6291 6293         PyObject *exc = NULL;
6292 6294
6293 6295         // so we can remember if we've seen an invalid escape char or not
6294 -         *first_invalid_escape = NULL;
6296 +         *first_invalid_escape_char = -1;
6297 +         *first_invalid_escape_ptr = NULL;
6295 6298
6296 6299         if (size == 0) {
6297 6300             if (consumed) {
↓... @@ -6474,9 +6477,12 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
↑... *s,
6474 6477         goto error;
6475 6478

```

```

6476 6479         default:
6477 -         if (*first_invalid_escape == NULL) {
6478 -             *first_invalid_escape = s-1; /* Back up one char, since we've
6479 -                 already incremented s. */
6480 +         if (*first_invalid_escape_char == -1) {
6481 +             *first_invalid_escape_char = c;
6482 +             if (starts == initial_starts) {
6483 +                 /* Back up one char, since we've already incremented s.
6484 +                 */
6485 +                 *first_invalid_escape_ptr = s - 1;
6486 +             }
6487         }
6488         WRITE_ASCII_CHAR('\\');
6489         WRITE_CHAR(c);
6490     @@ -6515,22 +6521,39 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
6515     *s,
6516     return NULL;
6517 }
6518
6519 + // Export for binary compatibility.
6520 + PyObject *
6521 + _PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
6522 +     Py_ssize_t size,
6523 +     const char *errors,
6524 +     Py_ssize_t *consumed,
6525 +     const char **first_invalid_escape)
6526 + {
6527 +     int first_invalid_escape_char;
6528 +     return _PyUnicode_DecodeUnicodeEscapeInternal2(
6529 +         s, size, errors, consumed,
6530 +         &first_invalid_escape_char,
6531 +         first_invalid_escape);
6532 + }
6533 +
6534 + PyObject *
6535 + _PyUnicode_DecodeUnicodeEscapeStateful(const char *s,
6536 +     Py_ssize_t size,
6537 +     const char *errors,
6538 +     Py_ssize_t *consumed)
6539 + {

```

```

6524 -     const char *first_invalid_escape;
6525 -     PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal(s, size,
        errors,
6545 +     int first_invalid_escape_char;
6546 +     const char *first_invalid_escape_ptr;
6547 +     PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal2(s, size,
        errors,
6526 6548                                     consumed,
6527 -                                     &first_invalid_escape);
6549 +
        &first_invalid_escape_char,
6550 +
        &first_invalid_escape_ptr);
6528 6551         if (result == NULL)
6529 6552             return NULL;
6530 -         if (first_invalid_escape != NULL) {
6553 +         if (first_invalid_escape_char != -1) {
6531 6554             if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
6532 6555                                     "invalid escape sequence '\\%c'",
6533 -                                     (unsigned char)*first_invalid_escape) < 0) {
6556 +                                     first_invalid_escape_char) < 0) {
6534 6557                 Py_DECREF(result);
6535 6558                 return NULL;
6536 6559             }

```

Parser/pegen/parse_string.c

```

@@ -119,12 +119,15 @@ decode_unicode_with_escapes(Parser *parser, const char
*s, size_t len, Token *t)
119 119     len = p - buf;
120 120     s = buf;
121 121
122 -     const char *first_invalid_escape;
123 -     v = _PyUnicode_DecodeUnicodeEscapeInternal(s, len, NULL, NULL,
        &first_invalid_escape);
124 -
125 -     if (v != NULL && first_invalid_escape != NULL) {
126 -         if (warn_invalid_escape_sequence(parser, *first_invalid_escape, t) < 0)
        {
127 -             /* We have not decref u before because first_invalid_escape points

```

```

122 +     int first_invalid_escape_char;
123 +     const char *first_invalid_escape_ptr;
124 +     v = _PyUnicode_DecodeUnicodeEscapeInternal2(s, (Py_ssize_t)len, NULL, NULL,
125 +                                               &first_invalid_escape_char,
126 +                                               &first_invalid_escape_ptr);
127 +
128 +     if (v != NULL && first_invalid_escape_ptr != NULL) {
129 +         if (warn_invalid_escape_sequence(parser, *first_invalid_escape_ptr, t)
130 + < 0) {
131 +             /* We have not decref u before because first_invalid_escape_ptr
132 + points
133 +         inside u. */
134 +             Py_XDECREF(u);
135 +             Py_DECREF(v);
136 +
137 + @@ -138,14 +141,17 @@ decode_unicode_with_escapes(Parser *parser, const char
138 + *s, size_t len, Token *t)
139 +
140 + static PyObject *
141 + decode_bytes_with_escapes(Parser *p, const char *s, Py_ssize_t len, Token *t)
142 + {
143 +     -     const char *first_invalid_escape;
144 +     -     PyObject *result = _PyBytes_DecodeEscape(s, len, NULL,
145 + &first_invalid_escape);
146 +
147 +     +     int first_invalid_escape_char;
148 +     +     const char *first_invalid_escape_ptr;
149 +     +     PyObject *result = _PyBytes_DecodeEscape2(s, len, NULL,
150 + &first_invalid_escape_char,
151 + &first_invalid_escape_ptr);
152 +
153 +     if (result == NULL) {
154 +         return NULL;
155 +     }
156 +
157 +     -     if (first_invalid_escape != NULL) {
158 +         -         if (warn_invalid_escape_sequence(p, *first_invalid_escape, t) < 0) {
159 +
160 +         +     if (first_invalid_escape_ptr != NULL) {
161 +         +         if (warn_invalid_escape_sequence(p, *first_invalid_escape_ptr, t) < 0)
162 +         {
163 +
164 +             Py_DECREF(result);
165 +             return NULL;
166 +         }
167 +     }
168 +
169 +
170 +
171 +
172 +
173 +
174 +
175 +
176 +
177 +
178 +
179 +
180 +
181 +
182 +
183 +
184 +
185 +
186 +
187 +
188 +
189 +
190 +
191 +
192 +
193 +
194 +
195 +
196 +
197 +
198 +
199 +
200 +
201 +
202 +
203 +
204 +
205 +
206 +
207 +
208 +
209 +
210 +
211 +
212 +
213 +
214 +
215 +
216 +
217 +
218 +
219 +
220 +
221 +
222 +
223 +
224 +
225 +
226 +
227 +
228 +
229 +
230 +
231 +
232 +
233 +
234 +
235 +
236 +
237 +
238 +
239 +
240 +
241 +
242 +
243 +
244 +
245 +
246 +
247 +
248 +
249 +
250 +
251 +
252 +
253 +
254 +
255 +
256 +
257 +
258 +
259 +
260 +
261 +
262 +
263 +
264 +
265 +
266 +
267 +
268 +
269 +
270 +
271 +
272 +
273 +
274 +
275 +
276 +
277 +
278 +
279 +
280 +
281 +
282 +
283 +
284 +
285 +
286 +
287 +
288 +
289 +
290 +
291 +
292 +
293 +
294 +
295 +
296 +
297 +
298 +
299 +
300 +
301 +
302 +
303 +
304 +
305 +
306 +
307 +
308 +
309 +
310 +
311 +
312 +
313 +
314 +
315 +
316 +
317 +
318 +
319 +
320 +
321 +
322 +
323 +
324 +
325 +
326 +
327 +
328 +
329 +
330 +
331 +
332 +
333 +
334 +
335 +
336 +
337 +
338 +
339 +
340 +
341 +
342 +
343 +
344 +
345 +
346 +
347 +
348 +
349 +
350 +
351 +
352 +
353 +
354 +
355 +
356 +
357 +
358 +
359 +
360 +
361 +
362 +
363 +
364 +
365 +
366 +
367 +
368 +
369 +
370 +
371 +
372 +
373 +
374 +
375 +
376 +
377 +
378 +
379 +
380 +
381 +
382 +
383 +
384 +
385 +
386 +
387 +
388 +
389 +
390 +
391 +
392 +
393 +
394 +
395 +
396 +
397 +
398 +
399 +
400 +
401 +
402 +
403 +
404 +
405 +
406 +
407 +
408 +
409 +
410 +
411 +
412 +
413 +
414 +
415 +
416 +
417 +
418 +
419 +
420 +
421 +
422 +
423 +
424 +
425 +
426 +
427 +
428 +
429 +
430 +
431 +
432 +
433 +
434 +
435 +
436 +
437 +
438 +
439 +
440 +
441 +
442 +
443 +
444 +
445 +
446 +
447 +
448 +
449 +
450 +
451 +
452 +
453 +
454 +
455 +
456 +
457 +
458 +
459 +
460 +
461 +
462 +
463 +
464 +
465 +
466 +
467 +
468 +
469 +
470 +
471 +
472 +
473 +
474 +
475 +
476 +
477 +
478 +
479 +
480 +
481 +
482 +
483 +
484 +
485 +
486 +
487 +
488 +
489 +
490 +
491 +
492 +
493 +
494 +
495 +
496 +
497 +
498 +
499 +
500 +
501 +
502 +
503 +
504 +
505 +
506 +
507 +
508 +
509 +
510 +
511 +
512 +
513 +
514 +
515 +
516 +
517 +
518 +
519 +
520 +
521 +
522 +
523 +
524 +
525 +
526 +
527 +
528 +
529 +
530 +
531 +
532 +
533 +
534 +
535 +
536 +
537 +
538 +
539 +
540 +
541 +
542 +
543 +
544 +
545 +
546 +
547 +
548 +
549 +
550 +
551 +
552 +
553 +
554 +
555 +
556 +
557 +
558 +
559 +
560 +
561 +
562 +
563 +
564 +
565 +
566 +
567 +
568 +
569 +
570 +
571 +
572 +
573 +
574 +
575 +
576 +
577 +
578 +
579 +
580 +
581 +
582 +
583 +
584 +
585 +
586 +
587 +
588 +
589 +
590 +
591 +
592 +
593 +
594 +
595 +
596 +
597 +
598 +
599 +
600 +
601 +
602 +
603 +
604 +
605 +
606 +
607 +
608 +
609 +
610 +
611 +
612 +
613 +
614 +
615 +
616 +
617 +
618 +
619 +
620 +
621 +
622 +
623 +
624 +
625 +
626 +
627 +
628 +
629 +
630 +
631 +
632 +
633 +
634 +
635 +
636 +
637 +
638 +
639 +
640 +
641 +
642 +
643 +
644 +
645 +
646 +
647 +
648 +
649 +
650 +
651 +
652 +
653 +
654 +
655 +
656 +
657 +
658 +
659 +
660 +
661 +
662 +
663 +
664 +
665 +
666 +
667 +
668 +
669 +
670 +
671 +
672 +
673 +
674 +
675 +
676 +
677 +
678 +
679 +
680 +
681 +
682 +
683 +
684 +
685 +
686 +
687 +
688 +
689 +
690 +
691 +
692 +
693 +
694 +
695 +
696 +
697 +
698 +
699 +
700 +
701 +
702 +
703 +
704 +
705 +
706 +
707 +
708 +
709 +
710 +
711 +
712 +
713 +
714 +
715 +
716 +
717 +
718 +
719 +
720 +
721 +
722 +
723 +
724 +
725 +
726 +
727 +
728 +
729 +
730 +
731 +
732 +
733 +
734 +
735 +
736 +
737 +
738 +
739 +
740 +
741 +
742 +
743 +
744 +
745 +
746 +
747 +
748 +
749 +
750 +
751 +
752 +
753 +
754 +
755 +
756 +
757 +
758 +
759 +
760 +
761 +
762 +
763 +
764 +
765 +
766 +
767 +
768 +
769 +
770 +
771 +
772 +
773 +
774 +
775 +
776 +
777 +
778 +
779 +
780 +
781 +
782 +
783 +
784 +
785 +
786 +
787 +
788 +
789 +
790 +
791 +
792 +
793 +
794 +
795 +
796 +
797 +
798 +
799 +
800 +
801 +
802 +
803 +
804 +
805 +
806 +
807 +
808 +
809 +
810 +
811 +
812 +
813 +
814 +
815 +
816 +
817 +
818 +
819 +
820 +
821 +
822 +
823 +
824 +
825 +
826 +
827 +
828 +
829 +
830 +
831 +
832 +
833 +
834 +
835 +
836 +
837 +
838 +
839 +
840 +
841 +
842 +
843 +
844 +
845 +
846 +
847 +
848 +
849 +
850 +
851 +
852 +
853 +
854 +
855 +
856 +
857 +
858 +
859 +
860 +
861 +
862 +
863 +
864 +
865 +
866 +
867 +
868 +
869 +
870 +
871 +
872 +
873 +
874 +
875 +
876 +
877 +
878 +
879 +
880 +
881 +
882 +
883 +
884 +
885 +
886 +
887 +
888 +
889 +
890 +
891 +
892 +
893 +
894 +
895 +
896 +
897 +
898 +
899 +
900 +
901 +
902 +
903 +
904 +
905 +
906 +
907 +
908 +
909 +
910 +
911 +
912 +
913 +
914 +
915 +
916 +
917 +
918 +
919 +
920 +
921 +
922 +
923 +
924 +
925 +
926 +
927 +
928 +
929 +
930 +
931 +
932 +
933 +
934 +
935 +
936 +
937 +
938 +
939 +
940 +
941 +
942 +
943 +
944 +
945 +
946 +
947 +
948 +
949 +
950 +
951 +
952 +
953 +
954 +
955 +
956 +
957 +
958 +
959 +
960 +
961 +
962 +
963 +
964 +
965 +
966 +
967 +
968 +
969 +
970 +
971 +
972 +
973 +
974 +
975 +
976 +
977 +
978 +
979 +
980 +
981 +
982 +
983 +
984 +
985 +
986 +
987 +
988 +
989 +
990 +
991 +
992 +
993 +
994 +
995 +
996 +
997 +
998 +
999 +
1000 +

```

Comments 0