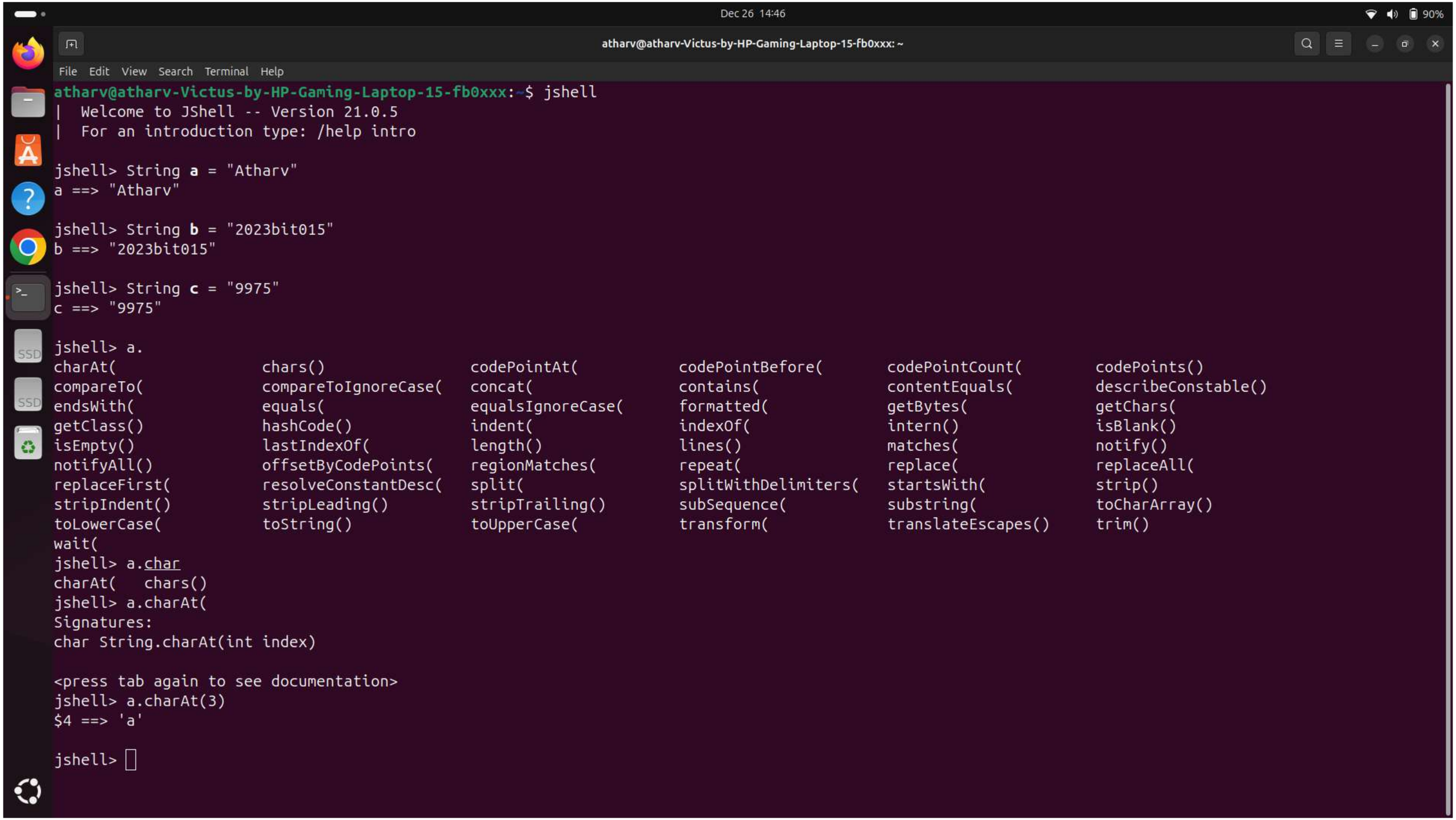
Dec 26 14:46
90%
atharv@atharv-Victus-by-HP-Gaming-Laptop-15-fb0xxx: ~
File Edit View Search Terminal Help
atharv@atharv-Victus-by-HP-Gaming-Laptop-15-fb0xxx:~$ jshell
| Welcome to JShell -- Version 21.0.5
| For an introduction type: /help intro
jshell> String a = "Atharv"
a ==> "Atharv"
jshell> String b = "2023bit015"
b ==> "2023bit015"
jshell> String c = "9975"
c ==> "9975"
jshell> a.
charAt(chars() codePointAt(codePointBefore(codePointCount(codePoints()
compareTo(compareToIgnoreCase(concat(contains(contentEquals(describeConstable()
endsWith(equals(equalsIgnoreCase(formatted(getBytes(getChars(
getClass() hashCode() indent(indexOf(intern() isBlank()
isEmpty() lastIndexOf(length() lines() matches(notify()
notifyAll() offsetByCodePoints(regionMatches(repeat(replace(replaceAll(
replaceFirst(resolveConstantDesc(split(splitWithDelimiters(startsWith(strip()
stripIndent() stripLeading() stripTrailing() subSequence(substring(toCharArray()
toLowerCase(toString() toUpperCase(transform(translateEscapes() trim()
wait(
jshell> a.char
charAt(chars()
jshell> a.charAt(
Signatures:
char String.charAt(int index)
<press tab again to see documentation>
jshell> a.charAt(3)
$4 ==> 'a'
jshell>

```
jshell> a.chars()
$6 ==> java.util.stream.IntPipeline$Head@6d4b1c02

jshell> c.codePointAt(
$4

Signatures:
int String.codePointAt(int index)

<press tab again to see documentation>
jshell> c.codePointAt(
int String.codePointAt(int index)
Returns the character (Unicode code point) at the specified index.The index refers to char
values (Unicode code units) and ranges from 0 to #length() - 1 .
If the char value specified at the given index is in the high-surrogate range, the following
index is less than the length of this String , and the char value at the following index is in
the low-surrogate range, then the supplementary code point corresponding to this surrogate pair
is returned. Otherwise, the char value at the given index is returned.

Parameters:
index - the index to the char values

Returns:
the code point value of the character at the index

Thrown Exceptions:
IndexOutOfBoundsException - if the index argument is negative or not less than the length of
                            this string.

<press tab again to see all possible completions; total possible completions: 558>
jshell> b.codePointAt(4)
$7 ==> 98

jshell> c.code
codePointAt(       codePointBefore(   codePointCount(    codePoints()
jshell> c.codePointAt(2)
$8 ==> 55
```

```
|        at String.codePointBefore (String.java:1617)
|        at (#12:1)

jshell> a.codePointBefore(
$10   $11   $12   $4    $7    $8    $9

Signatures:
int String.codePointBefore(int index)

<press tab again to see documentation>
jshell> a.codePointBefore(
int String.codePointBefore(int index)
Returns the character (Unicode code point) before the specified index.The index refers to char
values (Unicode code units) and ranges from 1 to length .
If the char value at (index - 1) is in the low-surrogate range, (index - 2) is not negative,
and the char value at (index - 2) is in the high-surrogate range, then the supplementary code
point value of the surrogate pair is returned. If the char value at index - 1 is an unpaired
low-surrogate or a high-surrogate, the surrogate value is returned.

Parameters:
index - the index following the code point that should be returned

Returns:
the Unicode code point value before the given index.

Thrown Exceptions:
IndexOutOfBoundsException - if the index argument is less than 1 or greater than the length of
                            this string.

<press tab again to see all possible completions; total possible completions: 564>
jshell> a.codePointBefore(1)
$13 ==> 65

jshell> b.codePointBefore(2)
$14 ==> 48

jshell>
```

```
                            this string.

<press tab again to see all possible completions; total possible completions: 569>
jshell> c.codePointCount(
$10   $11   $12   $13   $14   $15   $16   $17   $4    $7    $8    $9

Signatures:
int String.codePointCount(int beginIndex, int endIndex)

<press tab again to see documentation>
jshell> c.codePointCount(
int String.codePointCount(int beginIndex, int endIndex)
Returns the number of Unicode code points in the specified text range of this String .The text
range begins at the specified beginIndex and extends to the char at index endIndex - 1 . Thus
the length (in char s) of the text range is endIndex-beginIndex . Unpaired surrogates within
the text range count as one code point each.

Parameters:
beginIndex - the index to the first char of the text range.
endIndex - the index after the last char of the text range.

Returns:
the number of Unicode code points in the specified text range

Thrown Exceptions:
IndexOutOfBoundsException - if the beginIndex is negative, or endIndex is larger than the
                            length of this String , or beginIndex is larger than endIndex .

<press tab again to see all possible completions; total possible completions: 569>
jshell> c.codePointCount(2,4)
$18 ==> 2

jshell> a.codePointCount(2,6)
$19 ==> 4

jshell> a.codePoints()
$20 ==> java.util.stream.IntPipeline$Head@59e5ddf
```

```
string. The result is zero if the strings are equal; compareTo returns 0 exactly when the
#equals(Object) method would return true .
This is the definition of lexicographic ordering. If two strings are different, then either
they have different characters at some index that is a valid index for both strings, or their
lengths are different, or both. If they have different characters at one or more index
positions, let k be the smallest such index; then the string whose character at position k has
the smaller value, as determined by using the < operator, lexicographically precedes the other
string. In this case, compareTo returns the difference of the two character values at position
k in the two string -- that is, the value:
     this.charAt(k)-anotherString.charAt(k)

If there is no index position at which they differ, then the shorter string lexicographically
precedes the longer string. In this case, compareTo returns the difference of the lengths of
the strings -- that is, the value:
     this.length()-anotherString.length()

For finer-grained String comparison, refer to java.text.Collator .

Parameters:
anotherString - the String to be compared.

Returns:
the value 0 if the argument string is equal to this string; a value less than 0 if this string
is lexicographically less than the string argument; and a value greater than 0 if this string
is lexicographically greater than the string argument.

<press tab again to see all possible completions; total possible completions: 572>
jshell> a.compareTo("9975")
$21 ==> 8

jshell> a.compareTo(c)
$22 ==> 8

jshell> c.compareTo(a)
$23 ==> -8

jshell>
```

```
jshell> a.compareToIgnoreCase(
a   b   c

Signatures:
int String.compareToIgnoreCase(String str)

<press tab again to see documentation>
jshell> a.compareToIgnoreCase(
int String.compareToIgnoreCase(String str)
Compares two strings lexicographically, ignoring case differences.This method returns an
integer whose sign is that of calling compareTo with case folded versions of the strings where
case differences have been eliminated by calling
Character.toLowerCase(Character.toUpperCase(int)) on each Unicode code point.
Note that this method does not take locale into account, and will result in an unsatisfactory
ordering for certain locales. The java.text.Collator class provides locale-sensitive
comparison.

Parameters:
str - the String to be compared.

Returns:
a negative integer, zero, or a positive integer as the specified String is greater than, equal
to, or less than this String, ignoring case considerations.

<press tab again to see all possible completions; total possible completions: 575>
jshell> a.compareToIgnoreCase(c)
$24 ==> 40

jshell> a.compareToIgnoreCase(b)
$25 ==> 47

jshell> a.compareToIgnoreCase(a)
$26 ==> 0

jshell> c.compareToIgnoreCase(a)
$27 ==> -40
```

```
jshell> a.concat(
a   b   c

Signatures:
String String.concat(String str)

<press tab again to see documentation>
jshell> a.concat(
String String.concat(String str)
Concatenates the specified string to the end of this string.
If the length of the argument string is 0 , then this String object is returned. Otherwise, a
String object is returned that represents a character sequence that is the concatenation of the
character sequence represented by this String object and the character sequence represented by
the argument string.
Examples:
     "cares".concat("s") returns "caress"
     "to".concat("get").concat("her") returns "together"


Parameters:
str - the String that is concatenated to the end of this String .

Returns:
a string that represents the concatenation of this object's characters followed by the string
argument's characters.

<press tab again to see all possible completions; total possible completions: 579>
jshell> a.concat(c)
$28 ==> "Atharv9975"

jshell> a.concat(" Muttepawar")
$29 ==> "Atharv Muttepawar"

jshell> "Java".concat("Programming")
$30 ==> "JavaProgramming"

jshell>
```

```
jshell> a.contains(
$28   $29   $30   a     b     c

Signatures:
boolean String.contains(CharSequence s)

<press tab again to see documentation>
jshell> a.contains(
boolean String.contains(CharSequence s)
Returns true if and only if this string contains the specified sequence of char values.

Parameters:
s - the sequence to search for

Returns:
true if this string contains s , false otherwise

<press tab again to see all possible completions; total possible completions: 582>
jshell> b.contains(bit)
|  Error:
|  cannot find symbol
|    symbol:   variable bit
|  b.contains(bit)
|             ^-^

jshell> b.contains("bit")
$31 ==> true

jshell> a.contains("ha")
$32 ==> true

jshell> c.contains(a)
$33 ==> false

jshell> a.contains(a)
$34 ==> true
```

```
jshell> a.contentEquals(
boolean String.contentEquals(CharSequence cs)
Compares this string to the specified CharSequence .The result is true if and only if this
String represents the same sequence of char values as the specified sequence. Note that if the
CharSequence is a StringBuffer then the method synchronizes on it.
For finer-grained String comparison, refer to java.text.Collator .

Parameters:
cs - The sequence to compare this String against

Returns:
true if this String represents the same sequence of char values as the specified sequence,
false otherwise

<press tab again to see all possible completions; total possible completions: 586>
jshell> a.contentEquals("arv")
$35 ==> false

jshell> a.contentEquals("Atharv")
$36 ==> true

jshell> "bit ".contentEquals("bit")
$37 ==> false

jshell> b.contentEquals("bit")
$38 ==> false

jshell> a.describeConstable()
$39 ==> Optional[Atharv]

jshell> b.describeConstable()
$40 ==> Optional[2023bit015]

jshell> c.describeConstable()
$41 ==> Optional[9975]

jshell> a.describeConstable()
```

```
$41 ==> Optional[9975]

jshell> a.endsWith(
$28   $29   $30   a     b     c

Signatures:
boolean String.endsWith(String suffix)

<press tab again to see documentation>
jshell> a.endsWith(
boolean String.endsWith(String suffix)
Tests if this string ends with the specified suffix.

Parameters:
suffix - the suffix.

Returns:
true if the character sequence represented by the argument is a suffix of the character
sequence represented by this object; false otherwise. Note that the result will be true if the
argument is the empty string or is equal to this String object as determined by the
#equals(Object) method.

<press tab again to see all possible completions; total possible completions: 593>
jshell> a.endsWith(v)
|  Error:
|  cannot find symbol
|    symbol:   variable v
|  a.endsWith(v)
|             ^

jshell> a.endsWith("v")
$42 ==> true

jshell> b.endsWith("015")
$43 ==> true

jshell>
```

```
jshell> a.equals
equals(             equalsIgnoreCase(
jshell> a.equals(
$10   $11   $12   $13   $14   $15   $16   $17   $18   $19   $20   $21   $22   $23   $24   $25   $26   $27   $28   $29   $30   $31   $32   $33   $34
$35   $36   $37   $38   $39   $4    $40   $41   $42   $43   $5    $6    $7    $8    $9    a     b     c

Signatures:
boolean String.equals(Object anObject)

<press tab again to see documentation>
jshell> a.equals(
boolean String.equals(Object anObject)
Compares this string to the specified object.The result is true if and only if the argument is
not null and is a String object that represents the same sequence of characters as this object.
For finer-grained String comparison, refer to java.text.Collator .

Parameters:
anObject - The object to compare this String against

Returns:
true if the given object represents a String equivalent to this string, false otherwise

<press tab again to see all possible completions; total possible completions: 595>
jshell> a.equals(c)
$44 ==> false

jshell> a.equals("Atharv")
$45 ==> true

jshell> "2023bit015".equals(b)
$46 ==> true

jshell> a.equalsIgnoreCase("atHarV")
$47 ==> true

jshell>
```

```
<press tab again to see documentation>
jshell> a.formatted(
String String.formatted(Object...)
Formats using this string as the format string, and the supplied arguments.

Parameters:
args - Arguments referenced by the format specifiers in this string.

Returns:
A formatted string

<press tab again to see all possible completions; total possible completions: 599>
jshell> a.formatted("%s")
$48 ==> "Atharv"

jshell> a.formatted("%d")
$49 ==> "Atharv"

jshell> "%s".formatted(a)
$50 ==> "Atharv"

jshell> "%d".formatted(a)
|  Exception java.util.IllegalFormatConversionException: d != java.lang.String
|        at Formatter$FormatSpecifier.failConversion (Formatter.java:4515)
|        at Formatter$FormatSpecifier.printInteger (Formatter.java:3066)
|        at Formatter$FormatSpecifier.print (Formatter.java:3021)
|        at Formatter.format (Formatter.java:2791)
|        at Formatter.format (Formatter.java:2728)
|        at String.formatted (String.java:4452)
|        at (#51:1)

jshell> a.formatted("%c")
$52 ==> "Atharv"

jshell> c.formatted("%c")
$53 ==> "9975"
```

```
<press tab to see next documentation>
jshell> a.getBytes(
byte[] String.getBytes(java.nio.charset.Charset charset)
Encodes this String into a sequence of bytes using the given charset , storing the result into
a new byte array.
This method always replaces malformed-input and unmappable-character sequences with this
charset's default replacement byte array. The java.nio.charset.CharsetEncoder class should be
used when more control over the encoding process is required.

Parameters:
charset - The java.nio.charset.Charset to be used to encode the String

Returns:
The resultant byte array

<press tab to see next documentation>
jshell> a.getBytes(
byte[] String.getBytes()
Encodes this String into a sequence of bytes using the default charset , storing the result
into a new byte array.
The behavior of this method when this string cannot be encoded in the default charset is
unspecified. The java.nio.charset.CharsetEncoder class should be used when more control over
the encoding process is required.

Returns:
The resultant byte array

<press tab again to see all possible completions; total possible completions: 556>
jshell> a.getBytes()
$5 ==> byte[6] { 65, 116, 104, 97, 114, 118 }

jshell> b.getBytes()
$6 ==> byte[10] { 50, 48, 50, 51, 98, 105, 116, 48, 49, 53 }

jshell> c.getBytes()
$7 ==> byte[4] { 57, 57, 55, 53 }

jshell>
```

```
IndexOutOfBoundsException - If any of the following is true:
                            * srcBegin is negative.
                            * srcBegin is greater than srcEnd
                            * srcEnd is greater than the length of this string
                            * dstBegin is negative
                            * dstBegin+(srcEnd-srcBegin) is larger than dst.length

<press tab again to see all possible completions; total possible completions: 559>
jshell> a.getChars(1,4,char[] arr,1)
|  Error:
|  '.class' expected
|  a.getChars(1,4,char[] arr,1)
|                      ^

jshell> char[] arr = {'0','1'}
arr ==> char[2] { '0', '1' }

jshell> a.getChars(1,4,arr,1)
|  Exception java.lang.StringIndexOutOfBoundsException: Range [1, 1 + 3) out of bounds for length 2
|        at Preconditions$1.apply (Preconditions.java:55)
|        at Preconditions$1.apply (Preconditions.java:52)
|        at Preconditions$4.apply (Preconditions.java:213)
|        at Preconditions$4.apply (Preconditions.java:210)
|        at Preconditions.outOfBounds (Preconditions.java:98)
|        at Preconditions.outOfBoundsCheckFromIndexSize (Preconditions.java:118)
|        at Preconditions.checkFromIndexSize (Preconditions.java:397)
|        at String.checkBoundsOffCount (String.java:4853)
|        at String.getChars (String.java:1709)
|        at (#9:1)

jshell> a.getChars(1,2,arr,1)

jshell>

jshell> arr
arr ==> char[2] { '0', 't' }

jshell>
```

```
|  Exception java.lang.StringIndexOutOfBoundsException: Range [1, 1 + 3) out of bounds for length 2
|        at Preconditions$1.apply (Preconditions.java:55)
|        at Preconditions$1.apply (Preconditions.java:52)
|        at Preconditions$4.apply (Preconditions.java:213)
|        at Preconditions$4.apply (Preconditions.java:210)
|        at Preconditions.outOfBounds (Preconditions.java:98)
|        at Preconditions.outOfBoundsCheckFromIndexSize (Preconditions.java:118)
|        at Preconditions.checkFromIndexSize (Preconditions.java:397)
|        at String.checkBoundsOffCount (String.java:4853)
|        at String.getChars (String.java:1709)
|        at (#9:1)

jshell> a.getChars(1,2,arr,1)

jshell>

jshell> arr
arr ==> char[2] { '0', 't' }

jshell> a.getClass()
$12 ==> class java.lang.String

jshell> arr.getClass()
$13 ==> class [C

jshell> b.getClass()
$14 ==> class java.lang.String

jshell> a.hashCode()
$15 ==> 1971218384

jshell> b.hashCode()
$16 ==> 71139302

jshell> c.hashCode()
$17 ==> 1754622

jshell>
```

```
Signatures:
String String.indent(int n)

<press tab again to see documentation>
jshell> a.indent(
String String.indent(int n)
Adjusts the indentation of each line of this string based on the value of n , and normalizes
line termination characters.
This string is conceptually separated into lines using String#lines() . Each line is then
adjusted as described below and then suffixed with a line feed "\n" (U+000A). The resulting
lines are then concatenated and returned.
If n > 0 then n spaces (U+0020) are inserted at the beginning of each line.
If n < 0 then up to n  white space characters are removed from the beginning of each line. If a
given line does not contain sufficient white space then all leading white space characters are
removed. Each white space character is treated as a single character. In particular, the tab
character "\t" (U+0009) is considered a single character; it is not expanded.
If n == 0 then the line remains unchanged. However, line terminators are still normalized.

Parameters:
n - number of leading white space characters to add or remove

Returns:
string with indentation adjusted and line endings normalized

<press tab again to see all possible completions; total possible completions: 566>
jshell> a.indent(4)
$18 ==> "    Atharv\n"

jshell> b
b ==> "2023bit015"

jshell> b.indent(-4)
$20 ==> "2023bit015\n"

jshell> b.indent(4)
$21 ==> "    2023bit015\n"

jshell>
```

```
beginIndex - the index to start the search from (included).
endIndex - the index to stop the search at (excluded).

Returns:
the index of the first occurrence of the specified substring within the specified index range,
or -1 if there is no such occurrence.

Thrown Exceptions:
StringIndexOutOfBoundsException - if beginIndex is negative, or endIndex is larger than the
                                  length of this String object, or beginIndex is larger than
                                  endIndex .

<press tab again to see all possible completions; total possible completions: 569>
jshell> a.indexOf(5)
$22 ==> -1

jshell> a.indexOf("r")
$23 ==> 4

jshell> b.indexOf("bit")
$24 ==> 4

jshell> b.indexOf("bit",2)
$25 ==> 4

jshell> b.indexOf(1,2)
$26 ==> -1

jshell> c.indexOf(7)
$27 ==> -1

jshell> c.indexOf("7")
$28 ==> 2

jshell> a.indexOf("t",2,5)
$29 ==> -1

jshell>
```

```
jshell> a.indexOf(5)
$22 ==> -1

jshell> a.indexOf("r")
$23 ==> 4

jshell> b.indexOf("bit")
$24 ==> 4

jshell> b.indexOf("bit",2)
$25 ==> 4

jshell> b.indexOf(1,2)
$26 ==> -1

jshell> c.indexOf(7)
$27 ==> -1

jshell> c.indexOf("7")
$28 ==> 2

jshell> a.indexOf("t",2,5)
$29 ==> -1

jshell> a.intern()
$30 ==> "Atharv"

jshell> b.intern()
$31 ==> "2023bit015"

jshell> c.intern()
$32 ==> "9975"

jshell> a = c.intern()
a ==> "9975"

jshell> a
a ==> "9975"
```

```
jshell> a
a ==> "9975"

jshell> a.isBlank()
$35 ==> false

jshell> a.isEmpty()
$36 ==> false

jshell> d.isEmpty()
|  Error:
|  cannot find symbol
|    symbol:   variable d
|  d.isEmpty()
|  ^

jshell> String d
d ==> null

jshell> d.isEmpty()
|  Exception java.lang.NullPointerException: Cannot invoke "String.isEmpty()" because "REPL.$JShell$53.d" is null
|        at (#38:1)

jshell> d.isBlank()
|  Exception java.lang.NullPointerException: Cannot invoke "String.isBlank()" because "REPL.$JShell$53.d" is null
|        at (#39:1)

jshell> String d = ""
d ==> ""

jshell> d.isBlank()
$41 ==> true

jshell> d.isBlank()
$42 ==> true

jshell>
```

```
occurrence.

<press tab to see next documentation>
jshell> a.lastIndexOf(
int String.lastIndexOf(String str, int fromIndex)
Returns the index within this string of the last occurrence of the specified substring,
searching backward starting at the specified index.
The returned index is the largest value k for which:
     k <= Math.min(fromIndex, this.length()) &&
                   this.startsWith(str, k)

If no such value of k exists, then -1 is returned.

Parameters:
str - the substring to search for.
fromIndex - the index to start the search from.

Returns:
the index of the last occurrence of the specified substring, searching backward from the
specified index, or -1 if there is no such occurrence.

<press tab again to see all possible completions; total possible completions: 556>
jshell> a.lastIndexOf("r")
$5 ==> 4

jshell> b.lastIndexOf("bit")
$6 ==> 4

jshell> b.lastIndexOf("2024",1)
$7 ==> -1

jshell> b.lastIndexOf("2024",0)
$8 ==> -1

jshell> b.lastIndexOf("2024")
$9 ==> -1
```

```
jshell> a.length()
$4 ==> 6

jshell> b.length()
$5 ==> 10

jshell> c.length()
$6 ==> 8

jshell> a.lines()
$7 ==> java.util.stream.ReferencePipeline$Head@6093dd95

jshell> b.lines()
$8 ==> java.util.stream.ReferencePipeline$Head@4883b407

jshell> c.lines()
$9 ==> java.util.stream.ReferencePipeline$Head@39c0f4a

jshell> c.lines(
Signatures:
Stream<String> String.lines()

<press tab again to see documentation>
jshell> c.lines(
Stream<String> String.lines()
Returns a stream of lines extracted from this string, separated by line terminators.
A line terminator is one of the following: a line feed character "\n" (U+000A), a carriage
return character "\r" (U+000D), or a carriage return followed immediately by a line feed "\r\n"
(U+000D U+000A).
A line is either a sequence of zero or more characters followed by a line terminator, or it is
a sequence of one or more characters followed by the end of the string. A line does not include
the line terminator.
The stream returned by this method contains the lines from this string in the order in which
they occur.

Returns:
the stream of lines extracted from this string
```

```
|  <identifier> expected
|  java.util.stream.ReferencePipeline$Head@6093dd95
|                                          ^

jshell> a.matches(
a   b   c

Signatures:
boolean String.matches(String regex)

<press tab again to see documentation>
jshell> a.matches(
boolean String.matches(String regex)
Tells whether or not this string matches the given regular expression .
An invocation of this method of the form str .matches( regex ) yields exactly the same result
as the expression java.util.regex.Pattern . matches( regex , str )

Parameters:
regex - the regular expression to which this string is to be matched

Returns:
true if, and only if, this string matches the given regular expression

Thrown Exceptions:
PatternSyntaxException - if the regular expression's syntax is invalid

<press tab again to see all possible completions; total possible completions: 561>
jshell> a.matches("Atharv")
$10 ==> true

jshell> b.matches("bit")
$11 ==> false

jshell> b.matches(b)
$12 ==> true

jshell>
```

```
Returns:
true if, and only if, this string matches the given regular expression

Thrown Exceptions:
PatternSyntaxException - if the regular expression's syntax is invalid

<press tab again to see all possible completions; total possible completions: 564>
jshell> a.notify()
|  Exception java.lang.IllegalMonitorStateException: current thread is not owner
|        at Object.notify (Native Method)
|        at (#14:1)

jshell> a.notify(
Signatures:
void Object.notify()

<press tab again to see documentation>
jshell> a.notify(
void Object.notify()
<no documentation found>

<press tab again to see all possible completions; total possible completions: 564>
jshell> c.notif
notify()      notifyAll()
jshell> c.notify()
|  Exception java.lang.IllegalMonitorStateException: current thread is not owner
|        at Object.notify (Native Method)
|        at (#15:1)

jshell> c.notifyAll()
|  Exception java.lang.IllegalMonitorStateException: current thread is not owner
|        at Object.notifyAll (Native Method)
|        at (#16:1)

jshell> c.notifyAll(
Signatures:
void Object.notifyAll()
```

```
regionMatches(          repeat(             replace(            replaceAll(         replaceFirst(       resolveConstantDesc(  split(              splitWithDelimiters(
startsWith(             strip()             stripIndent()       stripLeading()      stripTrailing()     subSequence(          substring(          toCharArray()
toLowerCase(            toString()          toUpperCase(        transform(          translateEscapes()  trim()                wait(
jshell> a.offsetByCodePoints(
Signatures:
int String.offsetByCodePoints(int index, int codePointOffset)

<press tab again to see documentation>
jshell> a.offsetByCodePoints(
int String.offsetByCodePoints(int index, int codePointOffset)
Returns the index within this String that is offset from the given index by codePointOffset
code points.Unpaired surrogates within the text range given by index and codePointOffset count
as one code point each.

Parameters:
index - the index to be offset
codePointOffset - the offset in code points

Returns:
the index within this String

Thrown Exceptions:
IndexOutOfBoundsException - if index is negative or larger than the length of this String , or
                            if codePointOffset is positive and the substring starting with
                            index has fewer than codePointOffset code points, or if
                            codePointOffset is negative and the substring before index has
                            fewer than the absolute value of codePointOffset code points.

<press tab again to see all possible completions; total possible completions: 553>
jshell> a.offsetByCodePoints(1,2)
$2 ==> 3

jshell> a.offsetByCodePoints(3,1)
$3 ==> 4

jshell> a.offsetByCodePoints(3,4)
|  Exception java.lang.IndexOutOfBoundsException
|        at Character.offsetByCodePoints (Character.java:9750)
|        at String.offsetByCodePoints (String.java:1674)
|        at (#4:1)

jshell> a.offsetByCodePoints(3,3)
$5 ==> 6

jshell>
```

```
    all pairs of Unicode code points results in equal integer values.
Note that this method does not take locale into account, and will result in unsatisfactory
results for certain locales when ignoreCase is true . The java.text.Collator class provides
locale-sensitive comparison.

Parameters:
ignoreCase - if true , ignore case when comparing characters.
toffset - the starting offset of the subregion in this string.
other - the string argument.
ooffset - the starting offset of the subregion in the string argument.
len - the number of characters (Unicode code units - 16bit char value) to compare.

Returns:
true if the specified subregion of this string matches the specified subregion of the string
argument; false otherwise. Whether the matching is exact or case insensitive depends on the
ignoreCase argument.

<press tab again to see all possible completions; total possible completions: 563>
jshell> a.regionMatches(0,"Atharv",0,4)
$12 ==> true

jshell> a.regionMatches(0,"Atharv",1,4)
$13 ==> false

jshell> a.regionMatches(2,"Atharv",0,4)
$14 ==> false

jshell> a.regionMatches(0,"aThaRv",0,4)
$15 ==> false

jshell> a.regionMatches(true,0,"aThaRv",0,4)
$16 ==> true

jshell> a.regionMatches(2,"Atharv",2,4)
$17 ==> true

jshell>
```

```
jshell> a.regionMatches(2,"Atharv",2,4)
$17 ==> true

jshell> a.repeat(
Signatures:
String String.repeat(int count)

<press tab again to see documentation>
jshell> a.repeat(
String String.repeat(int count)
Returns a string whose value is the concatenation of this string repeated count times.
If this string is empty or count is zero then the empty string is returned.

Parameters:
count - number of times to repeat

Returns:
A string composed of this string repeated count times or the empty string if this string is
empty or count is zero

Thrown Exceptions:
IllegalArgumentException - if the count is negative.

<press tab again to see all possible completions; total possible completions: 569>
jshell> a.repeat(2)
$18 ==> "AtharvAtharv"

jshell> b.repeat(3)
$19 ==> "2023bit0152023bit0152023bit015"

jshell> c.repeat(0)
$20 ==> ""

jshell> c.repeat(1)
$21 ==> "98332155"

jshell>
```

```
string to the end, for example, replacing "aa" with "b" in the string "aaa" will result in "ba"
rather than "ab".

Parameters:
target - The sequence of char values to be replaced
replacement - The replacement sequence of char values

Returns:
The resulting string

<press tab again to see all possible completions; total possible completions: 573>
jshell> b.replace(23,24)
|  Error:
|  no suitable method found for replace(int,int)
|      method java.lang.String.replace(char,char) is not applicable
|        (argument mismatch; possible lossy conversion from int to char)
|      method java.lang.String.replace(java.lang.CharSequence,java.lang.CharSequence) is not applicable
|        (argument mismatch; int cannot be converted to java.lang.CharSequence)
|  b.replace(23,24)
|  ^-------^

jshell> b.replace("23","24")
$22 ==> "2024bit015"

jshell> b
b ==> "2023bit015"

jshell> a
a ==> "Atharv"

jshell> a.replace("a","_a")
$25 ==> "Ath_arv"

jshell> a
a ==> "Atharv"

jshell>
```

```
<press tab again to see documentation>
jshell> a.replaceAll(
String String.replaceAll(String regex, String replacement)
Replaces each substring of this string that matches the given regular expression with the given
replacement.
An invocation of this method of the form str .replaceAll( regex ,  repl ) yields exactly the
same result as the expression  java.util.regex.Pattern . compile ( regex ). matcher ( str ).
replaceAll ( repl )
Note that backslashes ( \ ) and dollar signs ( $ ) in the replacement string may cause the
results to be different than if it were being treated as a literal replacement string; see
Matcher.replaceAll . Use java.util.regex.Matcher#quoteReplacement to suppress the special
meaning of these characters, if desired.

Parameters:
regex - the regular expression to which this string is to be matched
replacement - the string to be substituted for each match

Returns:
The resulting String

Thrown Exceptions:
PatternSyntaxException - if the regular expression's syntax is invalid

<press tab again to see all possible completions; total possible completions: 555>
jshell> a.replaceAll(a,"sggs")
$4 ==> "sggs"

jshell> a
a ==> "Atharv"

jshell> a.replaceAll("arv","sggs")
$6 ==> "Athsggs"

jshell> b.replaceAll("bit","abcd")
$7 ==> "2023abcd015"

jshell>
```

```
same result as the expression  java.util.regex.Pattern . compile ( regex ). matcher ( str ).
replaceFirst ( repl )
Note that backslashes ( \ ) and dollar signs ( $ ) in the replacement string may cause the
results to be different than if it were being treated as a literal replacement string; see
java.util.regex.Matcher#replaceFirst . Use java.util.regex.Matcher#quoteReplacement to suppress
the special meaning of these characters, if desired.

Parameters:
regex - the regular expression to which this string is to be matched
replacement - the string to be substituted for the first match

Returns:
The resulting String

Thrown Exceptions:
PatternSyntaxException - if the regular expression's syntax is invalid

<press tab again to see all possible completions; total possible completions: 558>
jshell> a.replaceFirst(`
<press tab again to see all possible completions; total possible completions: 558>
jshell> a = "Atharv Atharv"
a ==> "Atharv Atharv"

jshell> a.replaceFirst(
$4   $6   $7   a    b    c

Signatures:
String String.replaceFirst(String regex, String replacement)

<press tab again to see documentation>
jshell> a.replaceFirst("Atharv", "Muttepawar")
$9 ==> "Muttepawar Atharv"

jshell> a
a ==> "Atharv Atharv"

jshell>
```

```
the String instance

<press tab again to see all possible completions; total possible completions: 559>
jshell> a.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup lookup)
|  Error:
|  ')' or ',' expected
|  a.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup lookup)
|                                                           ^

jshell> a.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup)
|  Error:
|  cannot find symbol
|    symbol:   variable Lookup
|  a.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup)
|                        ^-----------------------------------^

jshell> a.resolveConstantDesc(lookup)
|  Error:
|  cannot find symbol
|    symbol:   variable lookup
|  a.resolveConstantDesc(lookup)
|                        ^----^

jshell> a.resolveConstantDesc("lookup")
|  Error:
|  incompatible types: java.lang.String cannot be converted to java.lang.invoke.MethodHandles.Lookup
|  a.resolveConstantDesc("lookup")
|                        ^------^

jshell> a.resolveConstantDesc(MethodHandles.lookup())
|  Error:
|  cannot find symbol
|    symbol:   variable MethodHandles
|  a.resolveConstantDesc(MethodHandles.lookup())
|                        ^-----------^

jshell>
```

```
jshell> a.resolveConstantDesc(java.lang.invoke.MethodHandles.lookup())
$13 ==> "Atharv"

jshell> a.resolveConstantDesc(
Signatures:
String String.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup lookup)

<press tab again to see documentation>
jshell> a.resolveConstantDesc(
String String.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup lookup)
Resolves this instance as a ConstantDesc , the result of which is the instance itself.

Parameters:
lookup - ignored

Returns:
the String instance

<press tab again to see all possible completions; total possible completions: 560>
jshell> a.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup())
|  Error:
|  cannot find symbol
|    symbol:   method Lookup()
|  a.resolveConstantDesc(java.lang.invoke.MethodHandles.Lookup())
|                        ^---------------------------------^

jshell> a.resolveConstantDesc(java.lang.invoke.MethodHandles.lookup())
$14 ==> "Atharv"

jshell> b.resolveConstantDesc(java.lang.invoke.MethodHandles.lookup())
$15 ==> "2023bit015"

jshell> c.resolveConstantDesc(java.lang.invoke.MethodHandles.lookup())
$16 ==> "48622829"

jshell>
```

Split examples showing regex and result	**Regex**	**Result**
:	{ "boo", "and", "foo" }	
o	{ "b", "", ":and:f" }	

Parameters:
regex - the delimiting regular expression

Returns:
the array of strings computed by splitting this string around matches of the given regular expression

Thrown Exceptions:
PatternSyntaxException - if the regular expression's syntax is invalid

<press tab again to see all possible completions; total possible completions: 563>

```
jshell> a.split("a")
$17 ==> String[2] { "Ath", "rv" }

jshell> a.split("a",2)
$18 ==> String[2] { "Ath", "rv" }

jshell> a.split("a",1)
$19 ==> String[1] { "Atharv" }

jshell> a.split("a",3)
$20 ==> String[2] { "Ath", "rv" }

jshell> a.split("a",5)
$21 ==> String[2] { "Ath", "rv" }

jshell> a.split("t",2)
$22 ==> String[2] { "A", "harv" }
```

```
|                              | "foo" }                                |
-------------------------------------------------------------------------------------------
| o                            | 5                                      | { "b", "o", "", "o",          |
|                              |                                        | ":::and::f", "o", "", "o",    |
|                              |                                        | "" }                          |
-------------------------------------------------------------------------------------------
| -1                           | { "b", "o", "", "o",                   |
|                              | ":::and::f", "o", "", "o",             |
|                              | "" }                                   |
-------------------------------------------------------------------------------------------
| 0                            | { "b", "o", "", "o",                   |
|                              | ":::and::f", "o", "", "o" }            |
-------------------------------------------------------------------------------------------


Parameters:
regex - the delimiting regular expression
limit - the result threshold, as described above

Returns:
the array of strings computed by splitting this string around matches of the given regular
expression, alternating substrings and matching delimiters

<press tab again to see all possible completions; total possible completions: 569>
jshell> b.splitWithDelimiters("bit",2)
$23 ==> String[3] { "2023", "bit", "015" }

jshell> b.splitWithDelimiters("bit",3)
$24 ==> String[3] { "2023", "bit", "015" }

jshell> b.splitWithDelimiters("bit",0)
$25 ==> String[3] { "2023", "bit", "015" }

jshell> b.splitWithDelimiters("bit",-1)
$26 ==> String[3] { "2023", "bit", "015" }

jshell>
```

```
prefix - the prefix.
toffset - where to begin looking in this string.

Returns:
true if the character sequence represented by the argument is a prefix of the substring of this
         object starting at index toffset ; false otherwise. The result is false if toffset is
         negative or greater than the length of this String object; otherwise the result is
         the same as the result of the expression
                  this.substring(toffset).startsWith(prefix)


<press tab to see next documentation>
jshell> a.startsWith(
boolean String.startsWith(String prefix)
Tests if this string starts with the specified prefix.

Parameters:
prefix - the prefix.

Returns:
true if the character sequence represented by the argument is a prefix of the character
sequence represented by this string; false otherwise. Note also that true will be returned if
the argument is an empty string or is equal to this String object as determined by the
#equals(Object) method.

<press tab again to see all possible completions; total possible completions: 573>
jshell> a.startsWith("a")
$27 ==> false

jshell> a.startsWith("A")
$28 ==> true

jshell> b.startsWith("bit",4)
$29 ==> true

jshell> b.startsWith("bit")
$30 ==> false
```

```
jshell> b.startsWith("bit")
$30 ==> false

jshell> a.strip(
Signatures:
String String.strip()

<press tab again to see documentation>
jshell> a.strip(
String String.strip()
Returns a string whose value is this string, with all leading and trailing white space removed.
If this String object represents an empty string, or if all code points in this string are
white space , then an empty string is returned.
Otherwise, returns a substring of this string beginning with the first code point that is not a
white space up to and including the last code point that is not a white space .
This method may be used to strip white space from the beginning and end of a string.

Returns:
a string whose value is this string, with all leading and trailing white space removed

<press tab again to see all possible completions; total possible completions: 577>
jshell> a.strip()
$31 ==> "Atharv"

jshell> b = "2023 bit 015"
b ==> "2023 bit 015"

jshell> b.strip()
$33 ==> "2023 bit 015"

jshell> b = "  2023 bit 015"
b ==> "  2023 bit 015"

jshell> b.strip()
$35 ==> "2023 bit 015"

jshell>
```

```
First, the individual lines of this string are extracted. A line is a sequence of zero or more
characters followed by either a line terminator or the end of the string. If the string has at
least one line terminator, the last line consists of the characters between the last terminator
and the end of the string. Otherwise, if the string has no terminators, the last line is the
start of the string to the end of the string, in other words, the entire string. A line does
not include the line terminator.
Then, the minimum indentation (min) is determined as follows:
  * For each non-blank line (as defined by String#isBlank() ), the leading white space
    characters are counted.
  * The leading white space characters on the last line are also counted even if blank .
The min value is the smallest of these counts.

<press tab again to see next page>
jshell> b.stripIndent(
For each non-blank line, min leading white space characters are removed, and any trailing white
space characters are removed. Blank lines are replaced with the empty string.
Finally, the lines are joined into a new string, using the LF character "\n" (U+000A) to
separate lines.

Returns:
string with incidental indentation removed and line terminators normalized

<press tab again to see all possible completions; total possible completions: 580>
jshell> b.stripIndent()
$36 ==> "2023 bit 015"

jshell> b
b ==> "  2023 bit 015"

jshell> a = "   Atharv   Muttepawar  "
a ==> "   Atharv   Muttepawar  "

jshell> a.stripIndent()
$39 ==> "Atharv   Muttepawar"

jshell>
```

```
not include the line terminator.
Then, the minimum indentation (min) is determined as follows:
  * For each non-blank line (as defined by String#isBlank() ), the leading white space
    characters are counted.
  * The leading white space characters on the last line are also counted even if blank .
The min value is the smallest of these counts.

<press tab again to see next page>
jshell> b.stripIndent(
For each non-blank line, min leading white space characters are removed, and any trailing white
space characters are removed. Blank lines are replaced with the empty string.
Finally, the lines are joined into a new string, using the LF character "\n" (U+000A) to
separate lines.

Returns:
string with incidental indentation removed and line terminators normalized

<press tab again to see all possible completions; total possible completions: 580>
jshell> b.stripIndent()
$36 ==> "2023 bit 015"

jshell> b
b ==> "  2023 bit 015"

jshell> a = "   Atharv   Muttepawar  "
a ==> "   Atharv   Muttepawar  "

jshell> a.stripIndent()
$39 ==> "Atharv   Muttepawar"

jshell> a.stripLeading()
$40 ==> "Atharv   Muttepawar  "

jshell> a.stripTrailing()
$41 ==> "   Atharv   Muttepawar"

jshell>
```

```
$40 ==> "Atharv   Muttepawar  "

jshell> a.stripTrailing()
$41 ==> "   Atharv   Muttepawar"

jshell> a.subSequence(
Signatures:
CharSequence String.subSequence(int beginIndex, int endIndex)

<press tab again to see documentation>
jshell> a.subSequence(
CharSequence String.subSequence(int beginIndex, int endIndex)
Returns a character sequence that is a subsequence of this sequence.
An invocation of this method of the form
     str.subSequence(begin, end)
behaves in exactly the same way as the invocation
     str.substring(begin, end)

Parameters:
beginIndex - the begin index, inclusive.
endIndex - the end index, exclusive.

Returns:
the specified subsequence.

Thrown Exceptions:
IndexOutOfBoundsException - if beginIndex or endIndex is negative, if endIndex is greater than
                            length() , or if beginIndex is greater than endIndex

<press tab again to see all possible completions; total possible completions: 584>
jshell> a.subSequence(3,9)
$42 ==> "Atharv"

jshell> b.subSequence(7,10)
$43 ==> "bit"

jshell>
```

```
IndexOutOfBoundsException - if beginIndex is negative or larger than the length of this String
                            object.

<press tab to see next documentation>
jshell> a.substring(
String String.substring(int beginIndex, int endIndex)
Returns a string that is a substring of this string.The substring begins at the specified
beginIndex and extends to the character at index endIndex - 1 . Thus the length of the
substring is endIndex-beginIndex .
Examples:
     "hamburger".substring(4, 8) returns "urge"
     "smiles".substring(1, 5) returns "mile"


Parameters:
beginIndex - the beginning index, inclusive.
endIndex - the ending index, exclusive.

Returns:
the specified substring.

Thrown Exceptions:
IndexOutOfBoundsException - if the beginIndex is negative, or endIndex is larger than the
                            length of this String object, or beginIndex is larger than endIndex
                            .

<press tab again to see all possible completions; total possible completions: 587>
jshell> b.substring(3,7)
$45 ==> "023 "

jshell> b.substring(2,14)
$46 ==> "2023 bit 015"

jshell> b.substring(2,11)
$47 ==> "2023 bit "

jshell>
```

```
$47 ==> "2023 bit "

jshell> a.
charAt(                 chars()                 codePointAt(            codePointBefore(        codePointCount(         codePoints()
compareTo(              compareToIgnoreCase(    concat(                 contains(               contentEquals(          describeConstable()
endsWith(               equals(                 equalsIgnoreCase(       formatted(              getBytes(               getChars(
getClass()              hashCode()              indent(                 indexOf(                intern()                isBlank()
isEmpty()               lastIndexOf(            length()                lines()                 matches(                notify()
notifyAll()             offsetByCodePoints(     regionMatches(          repeat(                 replace(                replaceAll(
replaceFirst(           resolveConstantDesc(    split(                  splitWithDelimiters(    startsWith(             strip()
stripIndent()           stripLeading()          stripTrailing()         subSequence(            substring(              toCharArray()
toLowerCase(            toString()              toUpperCase(            transform(              translateEscapes()      trim()
wait(
jshell> a.toCharArray(
Signatures:
char[] String.toCharArray()

<press tab again to see documentation>
jshell> a.toCharArray(
char[] String.toCharArray()
Converts this string to a new character array.

Returns:
a newly allocated character array whose length is the length of this string and whose contents
are initialized to contain the character sequence represented by this string.

<press tab again to see all possible completions; total possible completions: 590>
jshell> a.toCharArray()
$48 ==> char[24] { ' ', ' ', ' ', 'A', 't', 'h', 'a', 'r', 'v', ' ', ' ', ' ', 'M', 'u', 't', 't', 'e', 'p', 'a', 'w', 'a', 'r', ' ', ' ' }

jshell> b.toCharArray()
$49 ==> char[14] { ' ', ' ', '2', '0', '2', '3', ' ', 'b', 'i', 't', ' ', '0', '1', '5' }

jshell> c.toCharArray()
$50 ==> char[8] { '4', '8', '6', '2', '2', '8', '2', '9' }

jshell>
```

			->small letter dotless i
(all)	**French Fries**	french fries	lowercased all chars in String
(all)	**IXΘΥΣ**	ιxθυσ	lowercased all chars in String

Parameters:

<press tab again to see next page>
jshell> a.toLowerCase(
locale - use the case transformation rules for this locale

Returns:
the String , converted to lowercase.

<press tab to see next documentation>
jshell> a.toLowerCase(
String String.toLowerCase()
Converts all of the characters in this String to lower case using the rules of the default
locale.This method is equivalent to toLowerCase(Locale.getDefault()) .

Returns:
the String , converted to lowercase.

<press tab again to see all possible completions; total possible completions: 594>
jshell> a.toLowerCase()
$52 ==> "   atharv   muttepawar "

jshell> c.toLowerCase()
$53 ==> "48622829"

jshell>

```
String String.toLowerCase()
Converts all of the characters in this String to lower case using the rules of the default
locale.This method is equivalent to toLowerCase(Locale.getDefault()) .

Returns:
the String , converted to lowercase.

<press tab again to see all possible completions; total possible completions: 594>
jshell> a.toLowerCase()
$52 ==> "   atharv   muttepawar  "

jshell> c.toLowerCase()
$53 ==> "48622829"

jshell> a.toString(
Signatures:
String String.toString()

<press tab again to see documentation>
jshell> a.toString(
String String.toString()
This object (which is already a string!) is itself returned.

Returns:
the string itself.

<press tab again to see all possible completions; total possible completions: 596>
jshell> a.toString()
$54 ==> "   Atharv   Muttepawar  "

jshell> b.toString()
$55 ==> "  2023 bit 015"

jshell> c.toString()
$56 ==> "48622829"

jshell>
```

tr (Turkish)	**\u0131**	\u0049	small letter dotless i ->capital letter I
(all)	**\u00df**	\u0053\u0053	small letter sharp s ->two letters: SS
(all)	**Fahrvergnügen**	FAHRVERGNÜGEN	

Parameters:
locale - use the case transformation rules for this locale

<press tab again to see next page>
jshell> a.toUpperCase(

Returns:
the String , converted to uppercase.

<press tab to see next documentation>
jshell> a.toUpperCase(
String String.toUpperCase()
Converts all of the characters in this String to upper case using the rules of the default
locale.This method is equivalent to toUpperCase(Locale.getDefault()) .

Returns:
the String , converted to uppercase.

<press tab again to see all possible completions; total possible completions: 599>
jshell> a.toUpperCase()
$57 ==> "   ATHARV   MUTTEPAWAR "

jshell> b.toUpperCase()
$58 ==> " 2023 BIT 015"

jshell>

```
R String.<R>transform(Function<? super String,? extends R> f)
This method allows the application of a function to this string.The function should expect a
single String argument and produce an R result.
Any exception thrown by f.apply() will be propagated to the caller.

Type Parameters:
R - the type of the result

Parameters:
f - a function to apply

Returns:
the result of applying the function to this string

<press tab again to see all possible completions; total possible completions: 601>
jshell> a.transform(toUpperCase())
|  Error:
|  cannot find symbol
|    symbol:   method toUpperCase()
|  a.transform(toUpperCase())
|              ^---------^

jshell> a.transform(b)
|  Error:
|  method transform in class java.lang.String cannot be applied to given types;
|    required: java.util.function.Function<? super java.lang.String,? extends R>
|    found:    java.lang.String
|    reason: cannot infer type-variable(s) R
|      (argument mismatch; java.lang.String cannot be converted to java.util.function.Function<? super java.lang.String,? extends R>)
|  a.transform(b)
|  ^---------^

jshell> a.transform(a -> a.toUpperCase())
$59 ==> "   ATHARV   MUTTEPAWAR  "

jshell> a.transform(b -> b.toUpperCase())
$60 ==> "   ATHARV   MUTTEPAWAR  "
```

\s	space	U+0020
\"	double quote	U+0022
\'	single quote	U+0027
****	backslash	U+005C
\0 - \377	octal escape	code point equivalents
\<line-terminator>	continuation	discard

Returns:

<press tab again to see next page>
jshell> a.translateEscapes(
String with escape sequences translated.

Thrown Exceptions:
IllegalArgumentException - when an escape sequence is malformed.

<press tab again to see all possible completions; total possible completions: 604>
jshell> a = "Atharv\tMuttepawar"
a ==> "Atharv\tMuttepawar"

jshell> a.translateEscapes()
$65 ==> "Atharv\tMuttepawar"

jshell> a = "Atharv\bMuttepawar"
a ==> "Atharv\bMuttepawar"

jshell> a.translateEscapes()
$67 ==> "Atharv\bMuttepawar"

jshell>

```
<press tab again to see next page>
jshell> a.translateEscapes(
String with escape sequences translated.

Thrown Exceptions:
IllegalArgumentException - when an escape sequence is malformed.

<press tab again to see all possible completions; total possible completions: 604>
jshell> a = "Atharv\tMuttepawar"
a ==> "Atharv\tMuttepawar"

jshell> a.translateEscapes()
$65 ==> "Atharv\tMuttepawar"

jshell> a = "Atharv\bMuttepawar"
a ==> "Atharv\bMuttepawar"

jshell> a.translateEscapes()
$67 ==> "Atharv\bMuttepawar"

jshell> a = "Atharv\tMuttepawar"
a ==> "Atharv\tMuttepawar"

jshell> a = "Atharv\\tMuttepawar"
a ==> "Atharv\\tMuttepawar"

jshell> a.translateEscapes()
$70 ==> "Atharv\tMuttepawar"

jshell> a = "Atharv\\nMuttepawar"
a ==> "Atharv\\nMuttepawar"

jshell> a.translateEscapes()
$72 ==> "Atharv\nMuttepawar"

jshell>
```

```
Signatures:
String String.trim()

<press tab again to see documentation>
jshell> a.trim(
String String.trim()
Returns a string whose value is this string, with all leading and trailing space removed, where
space is defined as any character whose codepoint is less than or equal to 'U+0020' (the space
character).
If this String object represents an empty character sequence, or the first and last characters
of character sequence represented by this String object both have codes that are not space (as
defined above), then a reference to this String object is returned.
Otherwise, if all characters in this string are space (as defined above), then a String object
representing an empty string is returned.
Otherwise, let k be the index of the first character in the string whose code is not a space
(as defined above) and let m be the index of the last character in the string whose code is not
a space (as defined above). A String object is returned, representing the substring of this
string that begins with the character at index k and ends with the character at index m -that
is, the result of this.substring(k, m + 1) .
This method may be used to trim space (as defined above) from the beginning and end of a
string.

Returns:
a string whose value is this string, with all leading and trailing space removed, or this
string if it has no leading or trailing space.

<press tab again to see all possible completions; total possible completions: 608>
jshell> b.trim()
$73 ==> "2023 bit 015"

jshell> a = "   Atharv   Muttepawar   "
a ==> "   Atharv   Muttepawar   "

jshell> a.trim()
$75 ==> "Atharv   Muttepawar"

jshell>
```

```
<press tab to see next documentation>
jshell> a.wait(
void Object.wait(long) throws InterruptedException
<no documentation found>

<press tab to see next documentation>
jshell> a.wait(
void Object.wait(long, int) throws InterruptedException
<no documentation found>

<press tab again to see all possible completions; total possible completions: 610>
jshell> a.wait()
|  Exception java.lang.IllegalMonitorStateException: current thread is not owner
|        at Object.wait0 (Native Method)
|        at Object.wait (Object.java:366)
|        at Object.wait (Object.java:339)
|        at (#76:1)

jshell> a.wait(123,4)
|  Exception java.lang.IllegalMonitorStateException: current thread is not owner
|        at Object.wait0 (Native Method)
|        at Object.wait (Object.java:366)
|        at Object.wait (Object.java:488)
|        at (#77:1)

jshell> b.wait()
|  Exception java.lang.IllegalMonitorStateException: current thread is not owner
|        at Object.wait0 (Native Method)
|        at Object.wait (Object.java:366)
|        at Object.wait (Object.java:339)
|        at (#78:1)

jshell> c.wait()
|  Exception java.lang.IllegalMonitorStateException: current thread is not owner
|        at Object.wait0 (Native Method)
|        at Object.wait (Object.java:366)
|        at Object.wait (Object.java:339)
|        at (#79:1)
```