Programming Lab

Autumn Semester

Course code: PC503

I can toast!
Let me rephrase that
I can do ANYTHING you tell me to...
... provided you speak my language

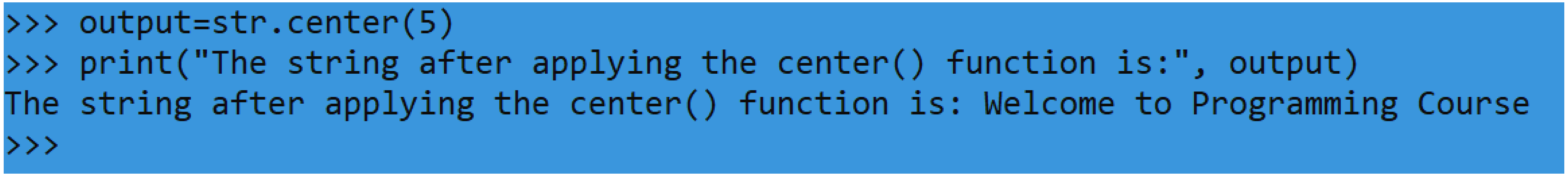
Dr. Rahul Mishra
Assistant Professor
DA-IICT, Gandhinagar

Lecture 4

Strings functions

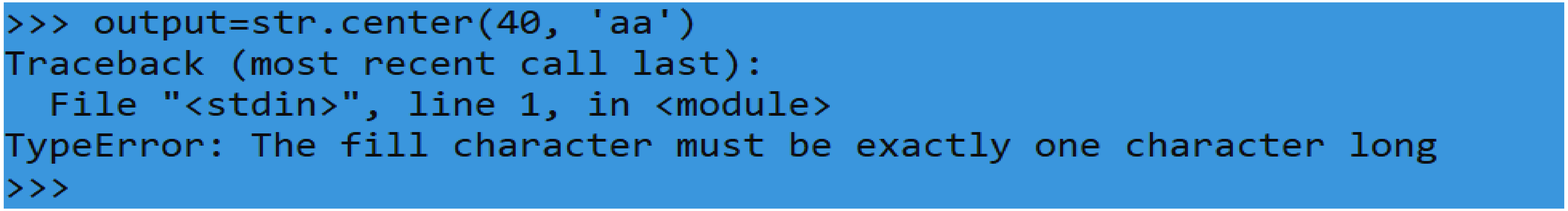
str.**center**(*width*[, *fillchar*])

- This function does not modify the original string if the parameter width that is mentioned is less than the length of the original input string.



```
>>> output=str.center(5)
>>> print("The string after applying the center() function is:", output)
The string after applying the center() function is: Welcome to Programming Course
>>>
```

- This function does not accept a string fillchar. It only accepts one character long fillchar. If the fillchar specified does not obey this condition, a type error is occured.



```
>>> output=str.center(40, 'aa')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: The fill character must be exactly one character long
>>>
```

str.count(sub[, start[, end]])

- Return the number of non-overlapping occurrences of substring sub in the range [start, end].

- Optional arguments' start and end is interpreted as in slice notation.

- If sub is empty, returns the number of empty strings between characters which is the length of the string plus one.

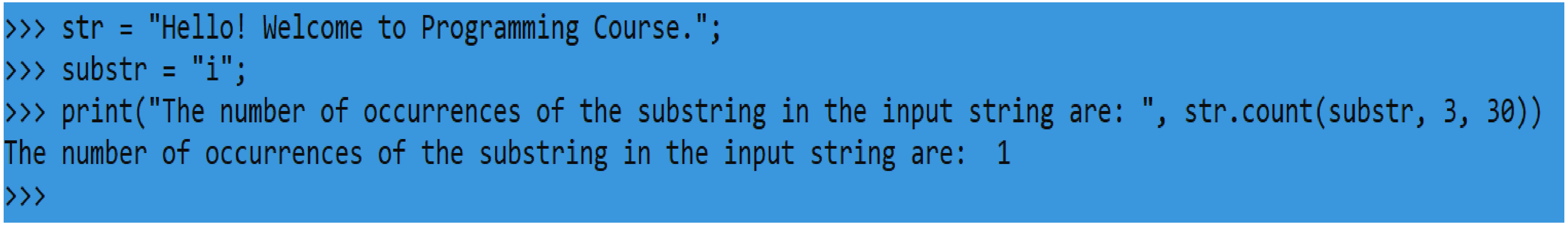
  - sub − This parameter specifies the substring to be searched.

  - start − This parameter is an integer value that specifies the starting index from which the search starts. The first character starts from the '0' index. If the start value is not specified, then the default value is '0' which is the first index.

  - end − This parameter is an integer value that specifies the ending index at which the search ends. If this value is not specified, then the default search ends at the last index.

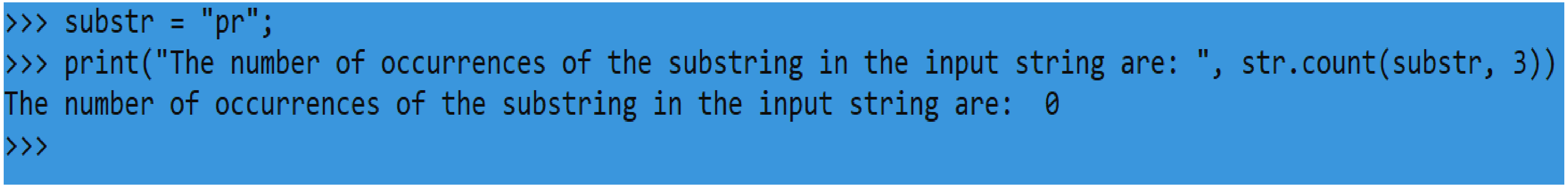
str.count(sub[, start[, end]])

- The python string count() method with a substring, start and end values as its parameters returns the count of number of occurrences of the substring within the specified range.



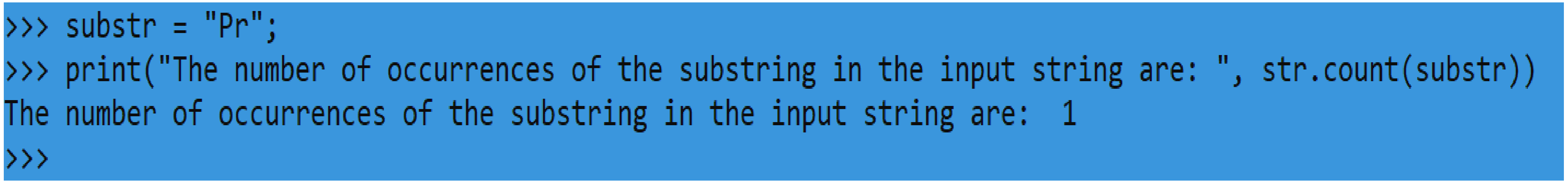
```
>>> str = "Hello! Welcome to Programming Course.";
>>> substr = "i";
>>> print("The number of occurrences of the substring in the input string are: ", str.count(substr, 3, 30))
The number of occurrences of the substring in the input string are:  1
>>>
```

- If the end value is not specified in the function's parameters, the last index of the string is considered as the default end value.



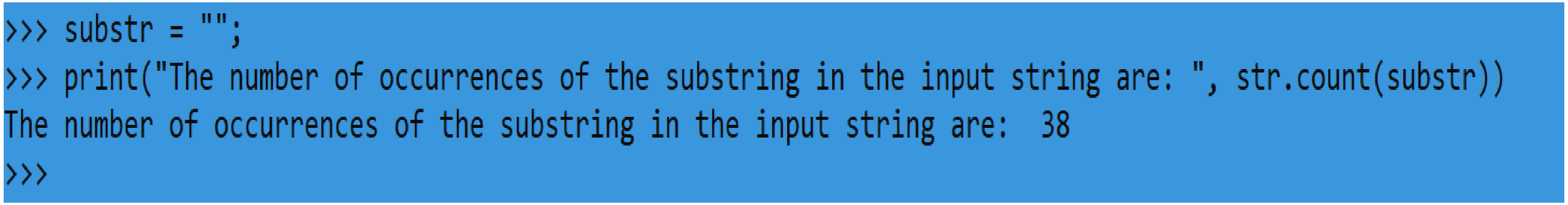
```
>>> substr = "pr";
>>> print("The number of occurrences of the substring in the input string are: ", str.count(substr, 3))
The number of occurrences of the substring in the input string are:  0
>>>
```

If the start and end values are not specified in the function's parameters, the zeroth index of the string is considered as the default start value and the last index of the string is considered as the default end value.



```
>>> substr = "Pr";
>>> print("The number of occurrences of the substring in the input string are: ", str.count(substr))
The number of occurrences of the substring in the input string are:  1
>>>
```

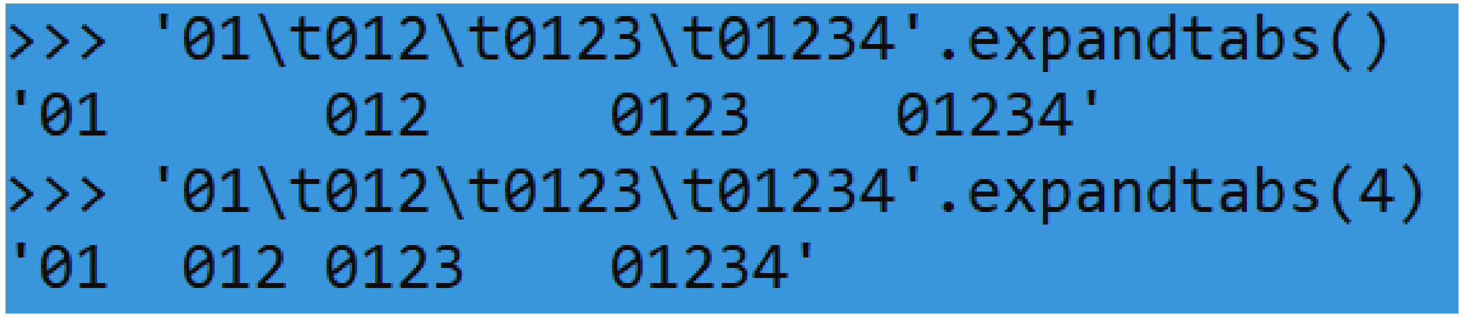
If the start and end values are not specified in the function's parameters, the zeroth index of the string is considered as the default start value and the last index of the string is considered as the default end value. If the sub string is empty, it returns the number of empty strings between characters which is the length of the string plus one.



```
>>> substr = "";
>>> print("The number of occurrences of the substring in the input string are: ", str.count(substr))
The number of occurrences of the substring in the input string are:  38
>>>
```

str.expandtabs(tabsize=8)

Return a copy of the string where all tab characters are replaced by one or more spaces, depending on the current column and the given tab size.

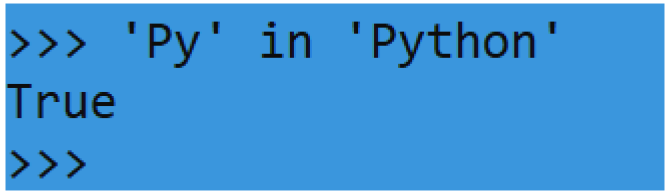


```
>>> '01\t012\t0123\t01234'.expandtabs()
'01      012     0123    01234'
>>> '01\t012\t0123\t01234'.expandtabs(4)
'01  012 0123    01234'
```

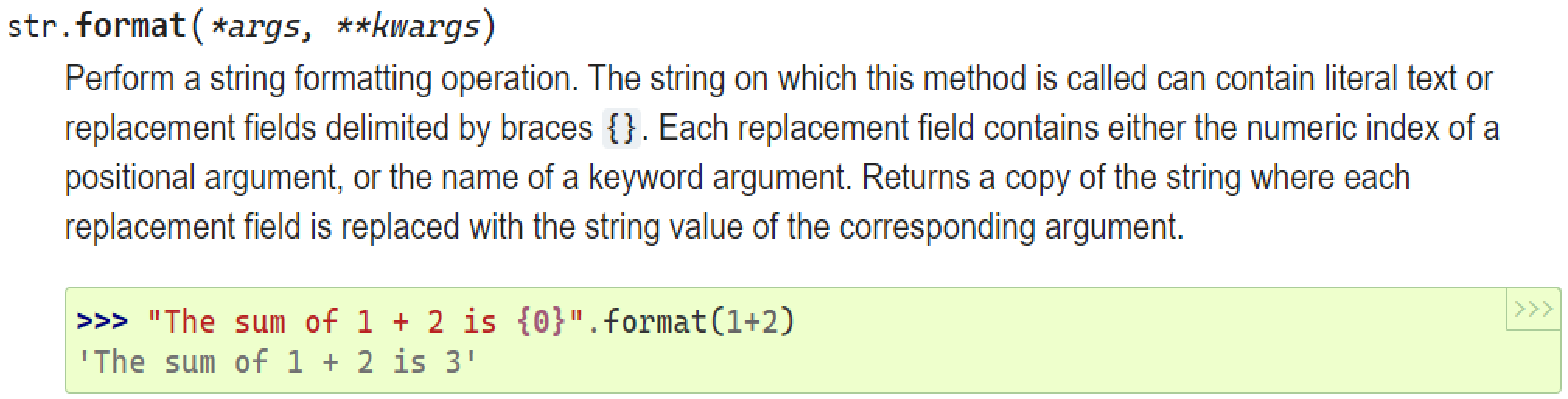
str.find(sub[, start[, end]])

Return the lowest index in the string where substring sub is found within the slice s[start:end]. Optional arguments start and end are interpreted as in slice notation. Return -1 if sub is not found.

The find() method should be used only if you need to know the position of sub. To check if sub is a substring or not, use the in operator



```
>>> 'Py' in 'Python'
True
>>>
```



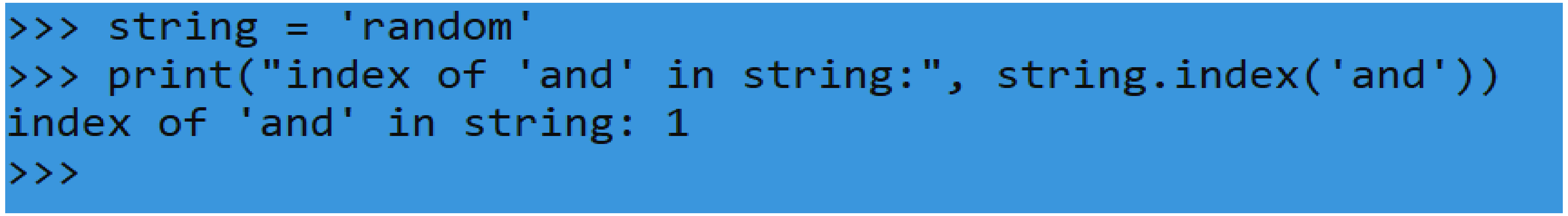
`str.format(*args, **kwargs)`

Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces `{}`. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. Returns a copy of the string where each replacement field is replaced with the string value of the corresponding argument.

```
>>> "The sum of 1 + 2 is {0}".format(1+2)
'The sum of 1 + 2 is 3'
```

str.index(sub[, start[, end]])

Like find(), but raise ValueError when the substring is not found



```
>>> string = 'random'
>>> print("index of 'and' in string:", string.index('and'))
index of 'and' in string: 1
>>>
```

Parameters:

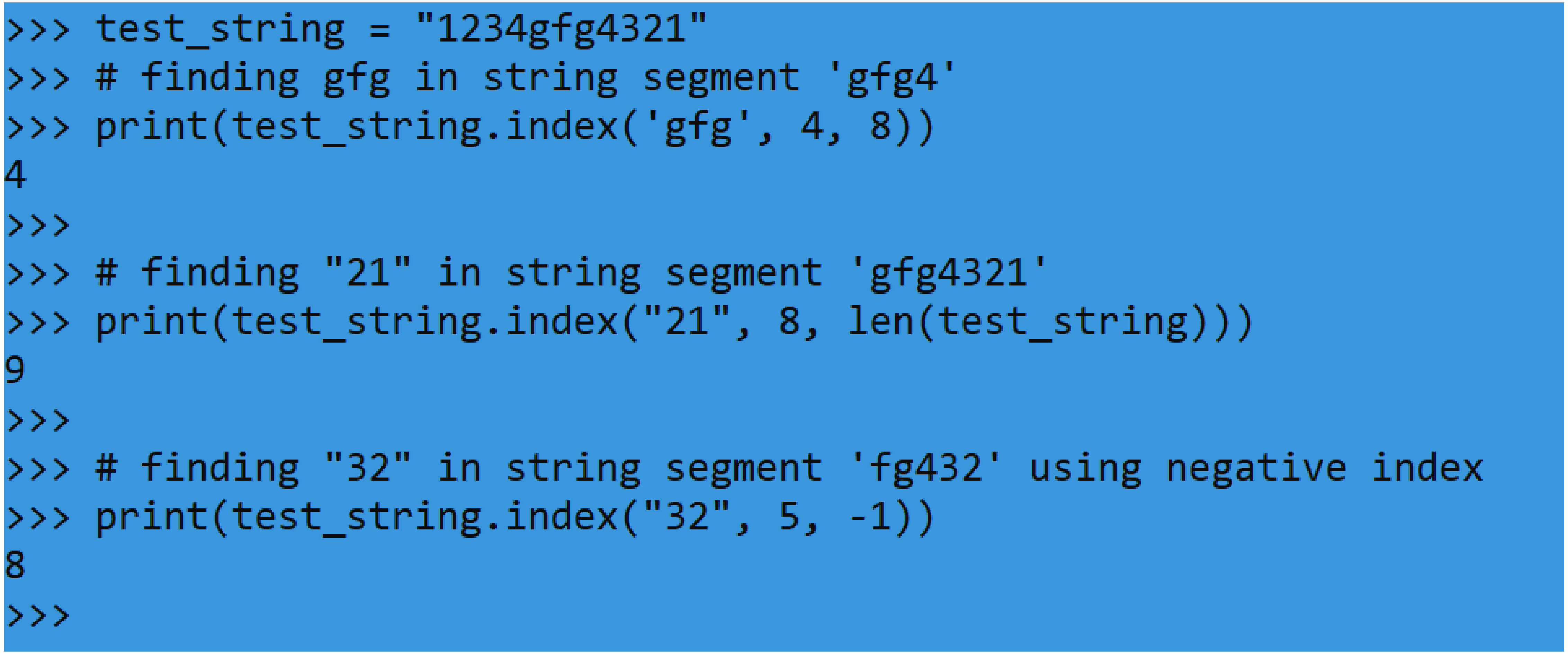
substring : The string to be searched for.

begp (default : 0) : This function specifies the position from where search has to be started.

endp (default: length of string): This function specifies the position from where search has to end.

Return: Returns the first position of the substring found.

Exception: Raises ValueError if argument string is not found or index is out of range.

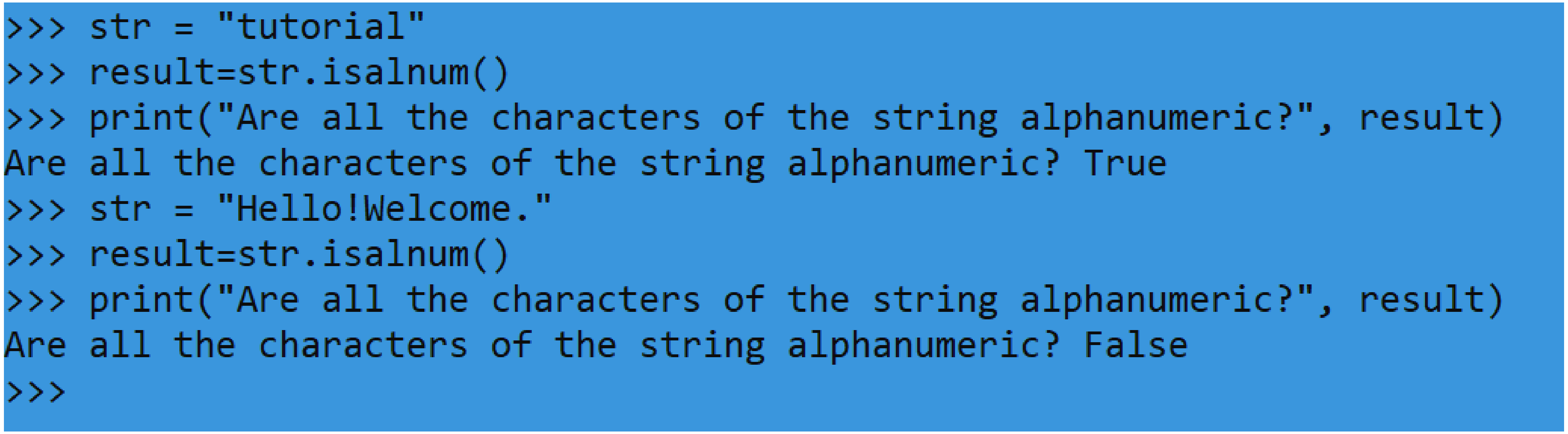


```
>>> test_string = "1234gfg4321"
>>> # finding gfg in string segment 'gfg4'
>>> print(test_string.index('gfg', 4, 8))
4
>>>
>>> # finding "21" in string segment 'gfg4321'
>>> print(test_string.index("21", 8, len(test_string)))
9
>>>
>>> # finding "32" in string segment 'fg432' using negative index
>>> print(test_string.index("32", 5, -1))
8
>>>
```

str.isalnum()

Return True if all characters in the string are alphanumeric and there is at least one character, False otherwise.

A character c is alphanumeric if one of the following returns True: **c.isalpha(), c.isdecimal(), c.isdigit(), or c.isnumeric().**



```
>>> str = "tutorial"
>>> result=str.isalnum()
>>> print("Are all the characters of the string alphanumeric?", result)
Are all the characters of the string alphanumeric? True
>>> str = "Hello!Welcome."
>>> result=str.isalnum()
>>> print("Are all the characters of the string alphanumeric?", result)
Are all the characters of the string alphanumeric? False
>>>
```

str.isalpha()

Return True if all characters in the string are alphabetic and there is at least one character, False otherwise. Alphabetic characters are those characters defined in the Unicode character database as “Letter”, i.e., those with general category property being one of “Lm”, “Lt”, “Lu”, “Ll”, or “Lo”.

str.isascii()

Return True if the string is empty or all characters in the string are ASCII, False otherwise.
ASCII characters have code points in the range U+0000-U+007F.

str.isdecimal()

Return True if all characters in the string are decimal characters and there is at least one character, False otherwise. Decimal characters are those that can be used to form numbers in base 10, e.g. U+0660, ARABIC-INDIC DIGIT ZERO. Formally a decimal character is a character in the Unicode General Category “Nd”.

Test all these functions with one or two examples

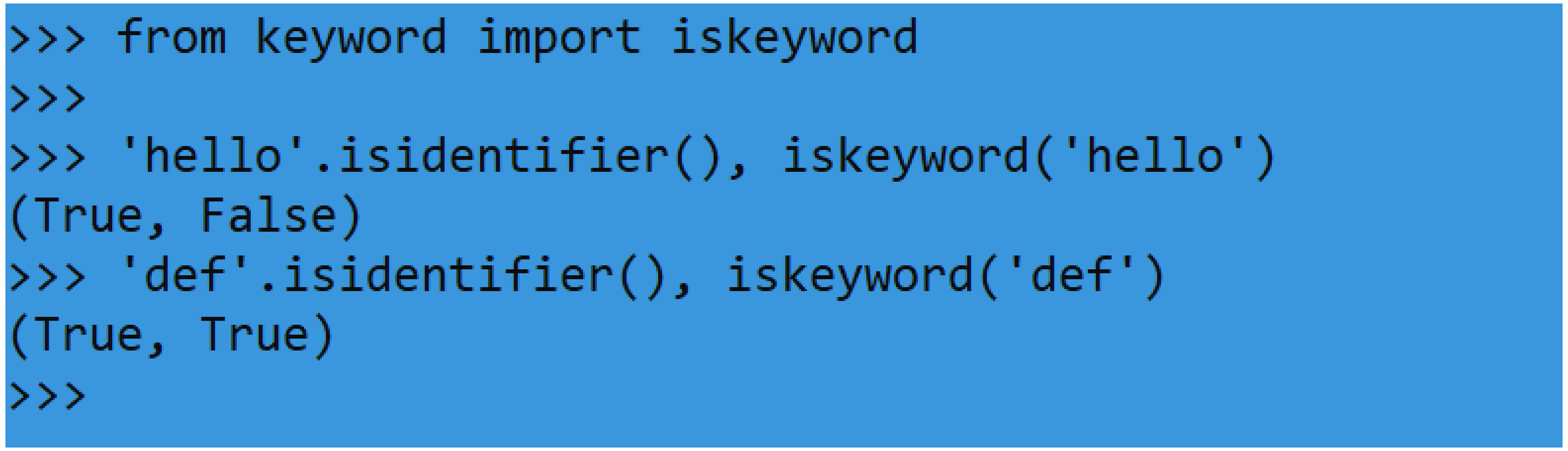
str.isdigit()

Return True if all characters in the string are digits and there is at least one character, False otherwise. Digits include decimal characters and digits that need special handling, such as the compatibility superscript digits. Formally, a digit is a character that has the property value Numeric_Type=Digit or Numeric_Type=Decimal.

str.isidentifier()

Return True if the string is a valid identifier according to the language definition, section Identifiers, and keywords.

Call **keyword.iskeyword()** to test whether string s is a reserved identifier, such as def and class.



```
>>> from keyword import iskeyword
>>>
>>> 'hello'.isidentifier(), iskeyword('hello')
(True, False)
>>> 'def'.isidentifier(), iskeyword('def')
(True, True)
>>>
```

str.islower()

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

str.isnumeric()

Return True if all characters in the string are numeric characters, and there is at least one character, False otherwise. Numeric characters include digit characters, and all characters that have the Unicode numeric value property, e.g. U+2155, VULGAR FRACTION ONE FIFTH. Formally, numeric characters are those with the property value Numeric_Type=Digit, Numeric_Type=Decimal or Numeric_Type=Numeric.

str.isprintable()

Return True if all characters in the string are printable or the string is empty, False otherwise. Nonprintable characters are those characters defined in the Unicode character database as “Other” or “Separator”, excepting the ASCII space (0x20) which is considered printable.

Test all these functions with one or two examples

str.isspace()

Return True if there are only whitespace characters in the string and there is at least one character, False otherwise.

A character is a whitespace if in the Unicode character database (see Unicode data), either its general category is Zs (“Separator, space”), or its bidirectional class is one of WS, B, or S.

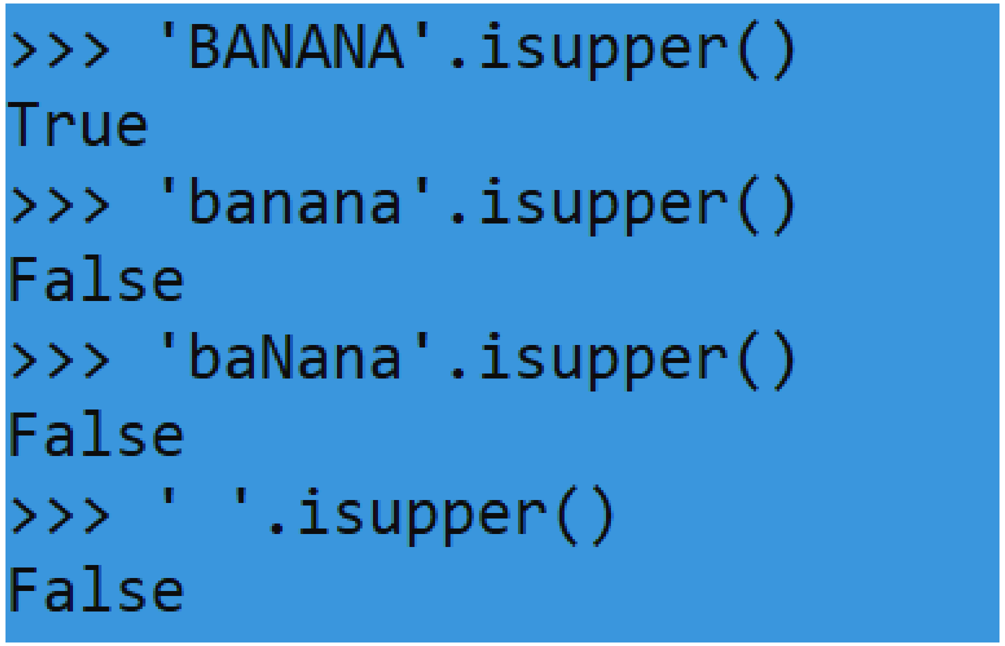
str.istitle()

Return True if the string is a title-cased string and there is at least one character, for example, uppercase characters may only follow uncased characters, and lowercase characters only cased ones. Return False otherwise.

str.isupper()

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

Test all these functions with one or two examples



```
>>> 'BANANA'.isupper()
True
>>> 'banana'.isupper()
False
>>> 'baNana'.isupper()
False
>>> ' '.isupper()
False
```

str.join(iterable)

Return a string which is the concatenation of the strings in iterable. A TypeError will be raised if there are any non-string values in iterable, including bytes objects. The separator between elements is the string providing this method.

str.ljust(width[, fillchar])

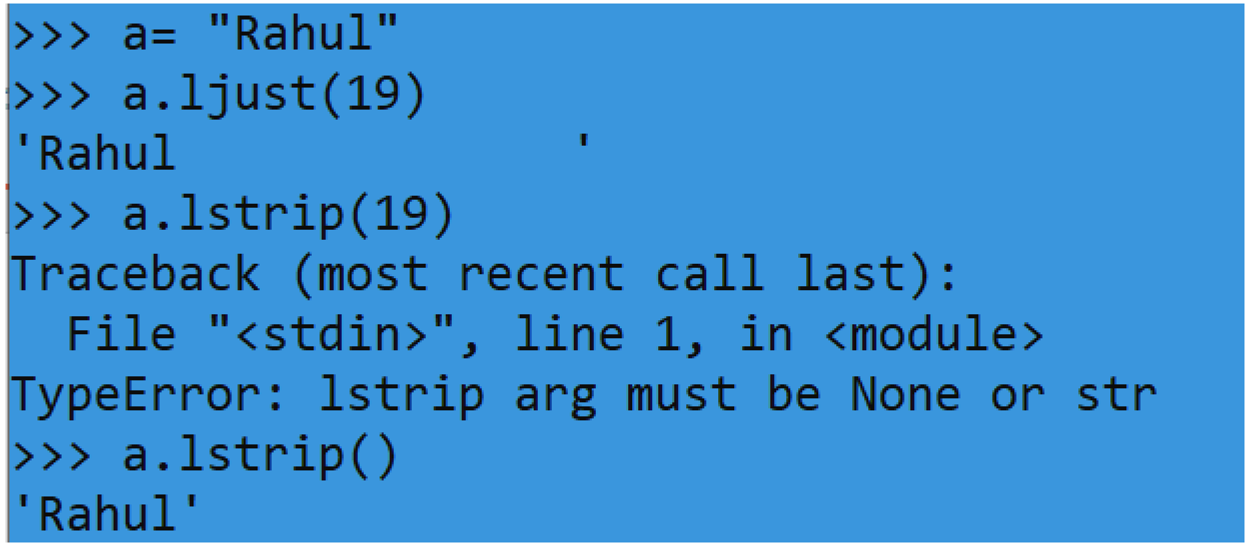
Return the string left justified in a string of length width. Padding is done using the specified fill char (default is an ASCII space). The original string is returned if the width is less than or equal to len(s).

str.lower()

Return a copy of the string with all the cased characters 4 converted to lowercase.

str.lstrip([chars])

Return a copy of the string with the leading characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument defaults to removing whitespace. The chars argument is not a prefix; rather, all combinations of its values are stripped.



```
>>> a= "Rahul"
>>> a.ljust(19)
'Rahul              '
>>> a.lstrip(19)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: lstrip arg must be None or str
>>> a.lstrip()
'Rahul'
```

`str.`**`replace`**`(`*`old, new`*`[, `*`count`*`])`

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

`str.`**`rfind`**`(`*`sub`*`[, `*`start`*`[, `*`end`*`]])`

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within `s[start:end]`. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

`str.`**`rindex`**`(`*`sub`*`[, `*`start`*`[, `*`end`*`]])`

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

`str.`**`rjust`**`(`*`width`*`[, `*`fillchar`*`])`

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

`str.rpartition(sep)`

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

`str.rsplit(sep=None, maxsplit=- 1)`

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.

`str.rstrip([chars])`

Return a copy of the string with trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

`str.strip([chars])`

Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

The outermost leading and trailing *chars* argument values are stripped from the string. Characters are removed from the leading end until reaching a string character that is not contained in the set of characters in *chars*. A similar action takes place on the trailing end. For example:

```
>>> comment_string = '#....... Section 3.2.1 Issue #32 .......'
>>> comment_string.strip('.#! ')
'Section 3.2.1 Issue #32'
```

`str.swapcase()`

Return a copy of the string with uppercase characters converted to lowercase and vice versa. Note that it is not necessarily true that `s.swapcase().swapcase() == s`.

Home assignment

Write three programs for each of the discussed string function.