Computing (ES 112)

Yogesh K. Meena
Shouvick Mondal

August 2024

Computer Science & Engineering
IIT Gandhinagar

Lecture 8

Strings

Strings in Python

- A String is a data type in Python that represents a sequence of characters. Examples: ‘IITGn’, “Hello”, “‘Hello World’”.
- It is an immutable data type, meaning that once you have created a string, you cannot change it.
- Usage: Storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.

```
>>> a = ‘IITGn’
>>> print(a)
'IITGn'
>>> a = “IITGn”
>>> print(a)
'IITGn'
>>> a = ‘‘‘IITGn’’’
>>> print(a)
'IITGn’
>>>a[0] = ‘B’
TypeError: ‘str’ object…
>>>a1 = ‘B’ + a[1:]
>>> print(a1)
BITGn
```

String Conversions: Recall

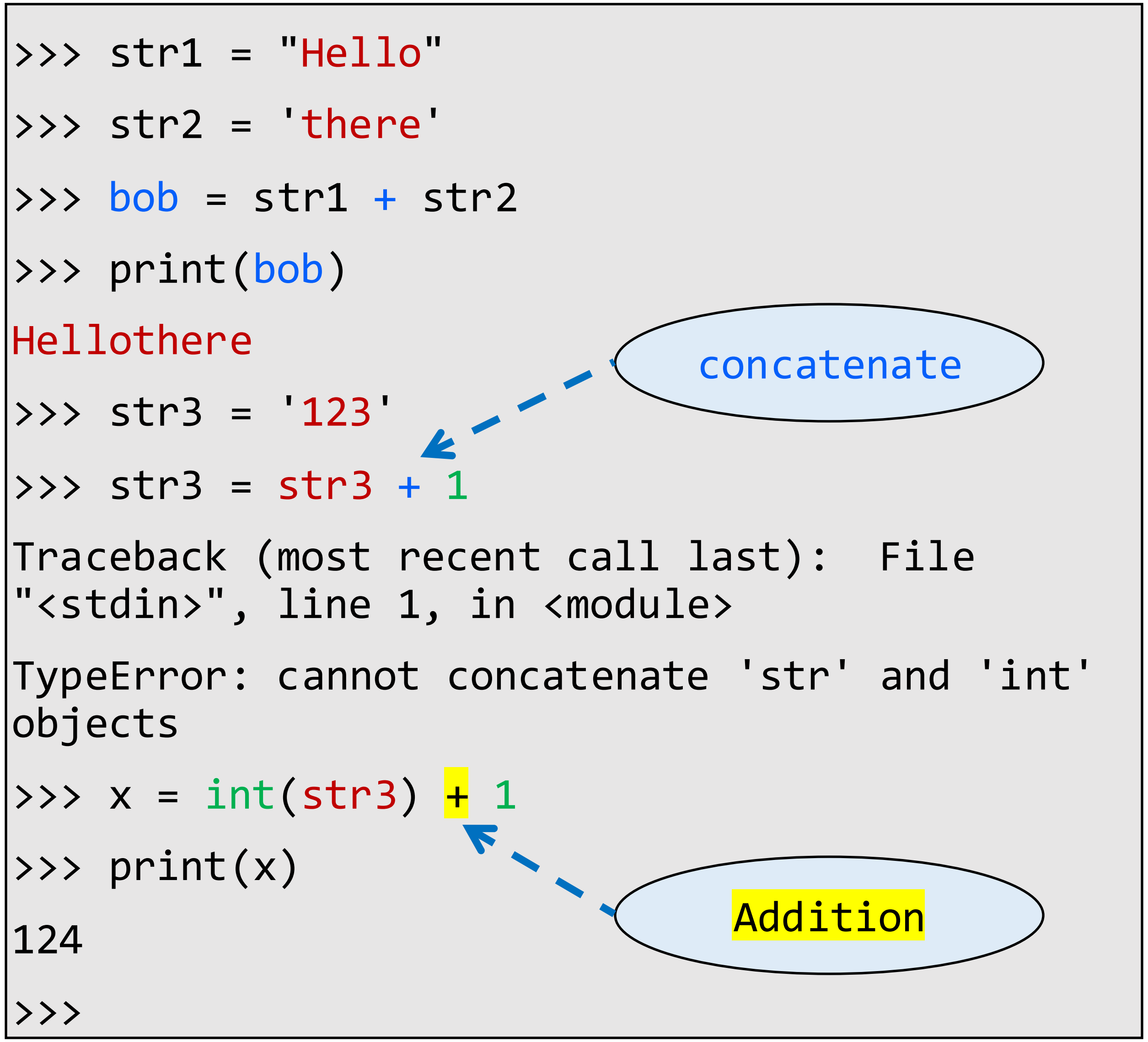
- You can also use int() and float() to convert between strings and integers
- You will get an **error** if the string does not contain numeric characters (also called as letters)

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'x'
```

How do string conversions affect the termination of indefinite loops accepting user input?

String Data Type

- A string is a sequence of characters
- A string literal uses quotes 'Hello' or "Hello"
- For strings, + means "concatenate"
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using int()



```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call last):  File
"<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int'
objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

Reading and Converting

- We prefer to read data in using strings and then parse and convert the data as we need
- This gives us more control over error situations and/or bad user input
- Input numbers must be converted from strings

```
>>> name = input('Enter:')
Enter:Chuck
>>> print(name)
Chuck
>>> apple = input('Enter:')
Enter:100
>>> x = apple – 10
Traceback (most recent call last):  File
"<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -:
'str' and 'int'
>>> x = int(apple) – 10
>>> print(x)
90
```

Looking Inside Strings

- We can get at any single character in a string using an index i specified in square brackets "[i]"
- The index value must be an integer and starts at zero
- The index value **can be** an expression that is computed

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

A Character Too Far, Out of Range!

- You will get a python error if you attempt to index beyond the end of a string
- So be careful when constructing index values and slices

```
>>> zot = 'abc'
>>> print(zot[5])
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
IndexError: string index out
of range
>>>
```

Strings Have Length

```
>>> fruit = 'banana'
>>> print(len(fruit))
6
>>> length = len(fruit)
>>> last = fruit[length]
IndexError: string index out of range
>>>last = fruit[length-1]
>>>print(last)
a
```

b	a	n	a	n	a
0	1	2	3	4	5

The built-in function `len` gives us the length of a string

Definite Loops: Iterating over characters in strings

```python
for i in "IIT GANDHINAGAR":
    print(i)
```

```
I
I
T
 
G
A
N
D
H
I
N
A
G
A
R
```

A string is a sequence of individual characters.

How can we print the characters on the same line with for loop?

```python
for i in "IIT GANDHINAGAR":
    print(i)
else:
    print("GUJARAT, INDIA")
```

```
I
I
T
 
G
A
N
D
H
I
N
A
G
A
R
GUJARAT, INDIA
```

"else:" can also be associated with a for loop.

Definite Loops: Iterating over characters in strings

```
for i in "IIT GANDHINAGAR":
    print(i,end="")
```

```
IIT GANDHINAGAR
```

A string is a sequence of individual characters.

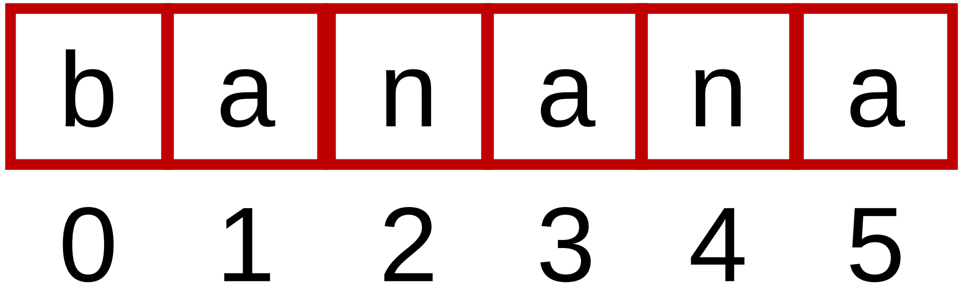
```
for i in "IIT GANDHINAGAR":
    print(i)
else:
    print("GUJARAT, INDIA")
```

```
I
I
T
 
G
A
N
D
H
I
N
A
G
A
R
GUJARAT, INDIA
```

"else:" can also be associated with a for loop.

Looping Through Strings

Using a while statement, an iteration variable, and the len function, we can construct a loop to look at each of the letters in a string individually



b	a	n	a	n	a
0	1	2	3	4	5

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

Output

```
0 b
1 a
2 n
3 a
4 n
5 a
```

Looping Through Strings

A definite loop using a for statement is much more elegant

The iteration variable is completely taken care of by the for loop

b	a	n	a	n	a
0	1	2	3	4	5

```
fruit = 'banana'
for letter in fruit:
    print(letter)
```

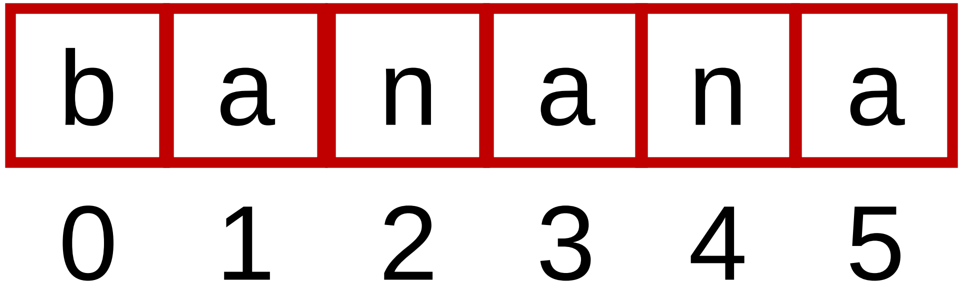
Output

```
b
a
n
a
n
a
```

Looping and counting

This is a simple loop that loops through each letter in a string and counts the number of times the loop encounters the 'a' character

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
print(count)
```



b	a	n	a	n	a
0	1	2	3	4	5

Output

```
3
```

String index values can be negative

```
fruit = 'banana'
for i in range(0,len(fruit)):
    print(i,fruit[i])
```

Output

```
0 b
1 a
2 n
3 a
4 n
5 a
```

b	a	n	a	n	a
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

```
fruit = 'banana'
for i in range(-len(fruit),0,1):
    print(i,fruit[i])
```

Output

```
-6 b
-5 a
-4 n
-3 a
-2 n
-1 a
```

Indexing is for extracting a single item from an existing index (within bounds)

Slicing Strings in Python

I	I	T		G	A	N	D	H	I	N	A	G	A	R
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

- We can also look at any continuous section of a string using a colon operator
- The **second number** is one beyond the end of the slice - “up to but not including”
- If the **second number** is beyond the end of the string, it stops at the end

`S[start:stop:step]`

Slicing Strings in Python

I	I	T		G	A	N	D	H	I	N	A	G	A	R
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

`S[start:stop:step]`

Similar to `range(start,stop,step)` for integers

Extract the characters in sequence which

- (optional) Starts at position (`start`)
- (optional) Ends at position (`stop-1`)
- (optional) Step-through in steps of size (`step`)

On omission, **start** is 0

On omission, **stop** is `len(S)-1`

On omission, **step** is 1

Slicing is for extracting a subsequence (on success) from a sequence, and empty '' (on failure).

Slicing Strings in Python

I	I	T		G	A	N	D	H	I	N	A	G	A	R
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

S[start:stop:step]

What will be value of S[::-1]?

```
>>>S='IIT-GANDHINAGAR'
>>>print(S[0:4])
IIT-
>>> print(S[2:6])
T-GA
```

```
>>> print(S[:6])
IIT-GA
>>> print(S[6:])
NDHINAGAR
>>> print(S[6:6])

```

Empty string '' with length 0

Using in a logical operator

- The in keyword can also be used to check to see if one string is “in” another string
- The in expression is a logical expression that returns True or False and can be used in an if statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
```

String Comparison

String comparison is Python is **lexicographical** which means dictionary order

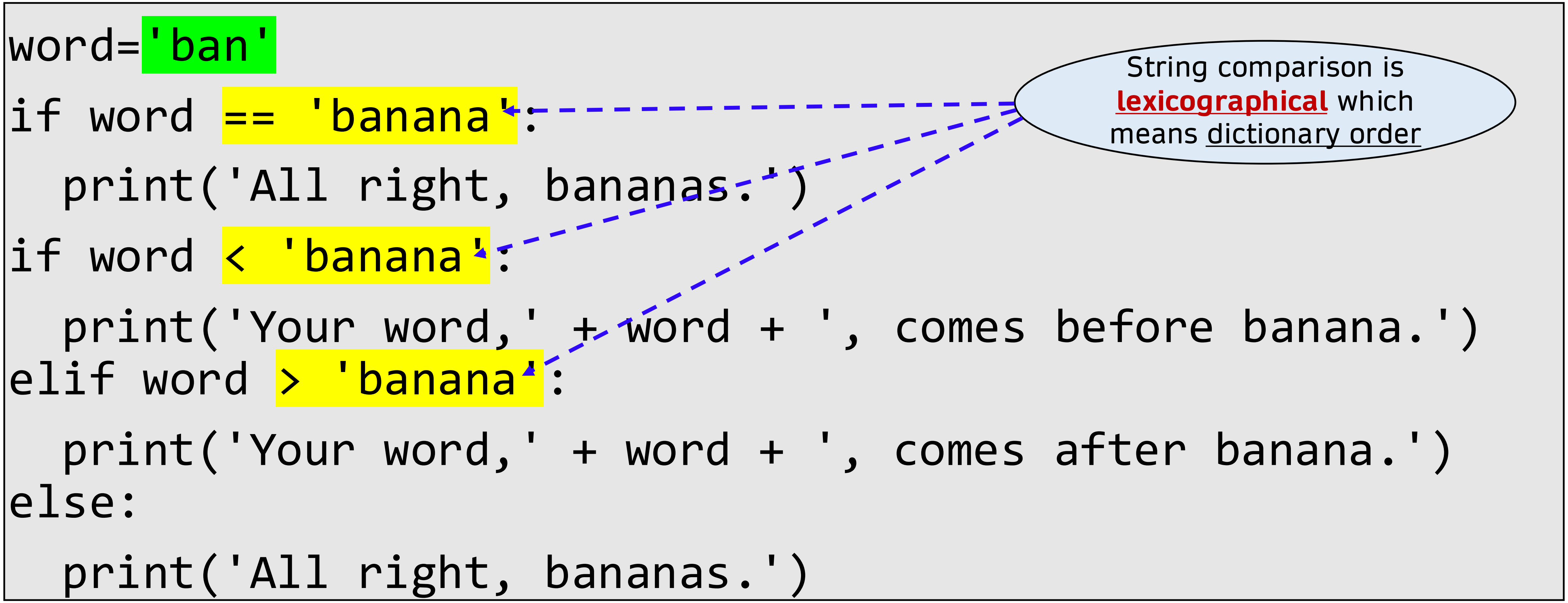
If len(s1) < len(s2), then it may not be the case that s1 < s2 (comes before)

```
s1 < s2 if
    1. ord(s1[0]) < ord(s2[0]) or
    2. ord(s1[0]) == ord(s2[0]) and ord(s1[1]) < ord(s2[1]) or
    3. ord(s1[0]) == ord(s2[0]) and ord(s1[1]) == ord(s2[1]) and ord(s1[2]) < ord(s2[2]) or
    •  ........................................... s1 exhausts before s2
```

```
ord(' ') = 32
ord('0') = 48
ord('A') = 65
ord('a') = 97
```

The empty string '' comes before any other string in the dictionary order.

String Comparison



```
word='ban'
if word == 'banana':
  print('All right, bananas.')
if word < 'banana':
  print('Your word,' + word + ', comes before banana.')
elif word > 'banana':
  print('Your word,' + word + ', comes after banana.')
else:
  print('All right, bananas.')
```

String comparison is **lexicographical** which means dictionary order

```
Your word, ban, comes before banana.
```

String Library

I	I	T		G	A	N	D	H	I	N	A	G	A	R
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- Python has a number of string functions which are in the string library
- These functions are already built into every string - we invoke (call) them by appending the function to the string variable
- These functions do not modify the original string, instead they return a new string that has been altered

https://docs.python.org/3.10/library/string.html

String Library

I	I	T		G	A	N	D	H	I	N	A	G	A	R
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

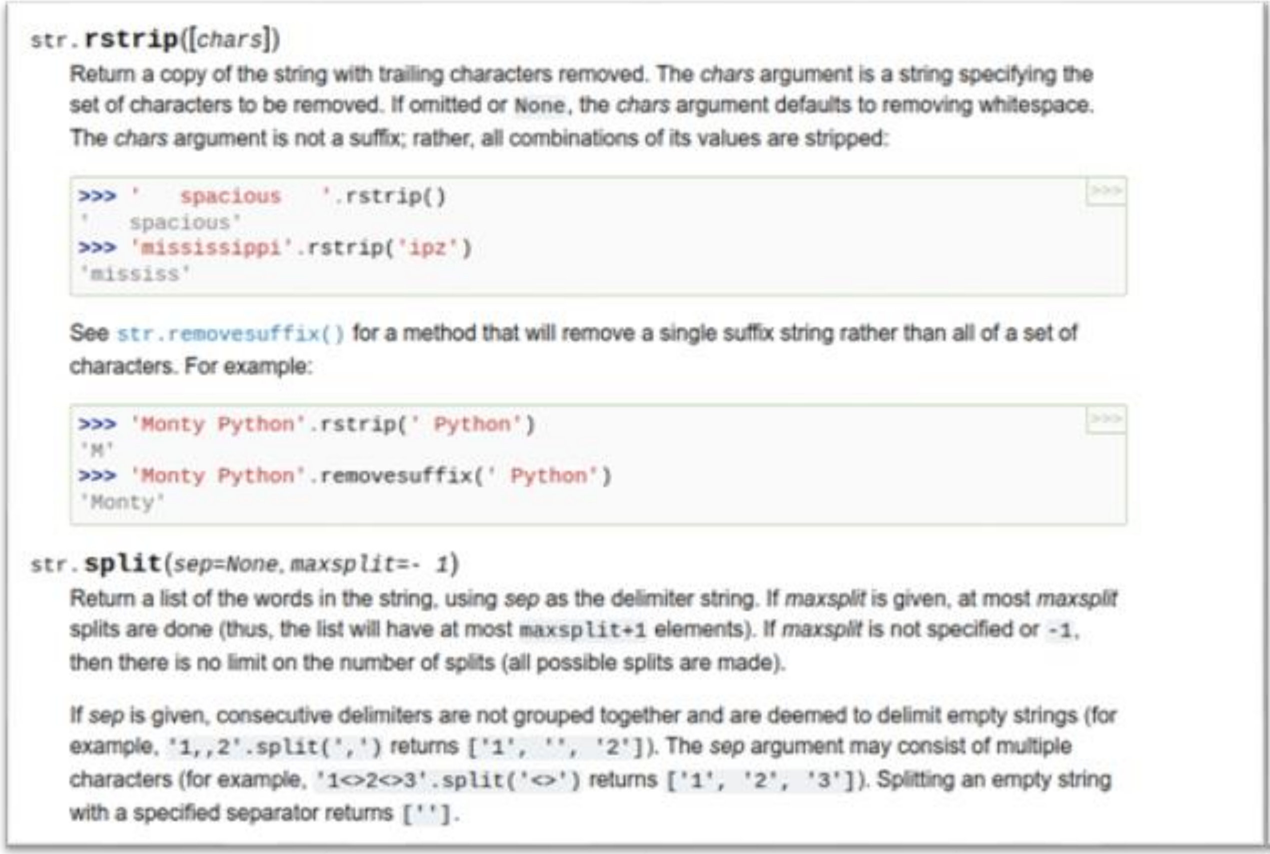
```
>>> S='IIT GANDHINAGAR'
>>> print(S.lower())
iit gandhinagar
>>> print(S.upper())
IIT GANDHINAGAR
```

```
>>> dir(S)
['__add__', '__class__', '__contains__', '__delattr__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__',
'__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__',
'__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal',
'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join',
'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
'removeprefix', 'removesuffix', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

All possible string functions...

https://docs.python.org/3.10/library/stdtypes.html#string-methods

String Library



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'
```

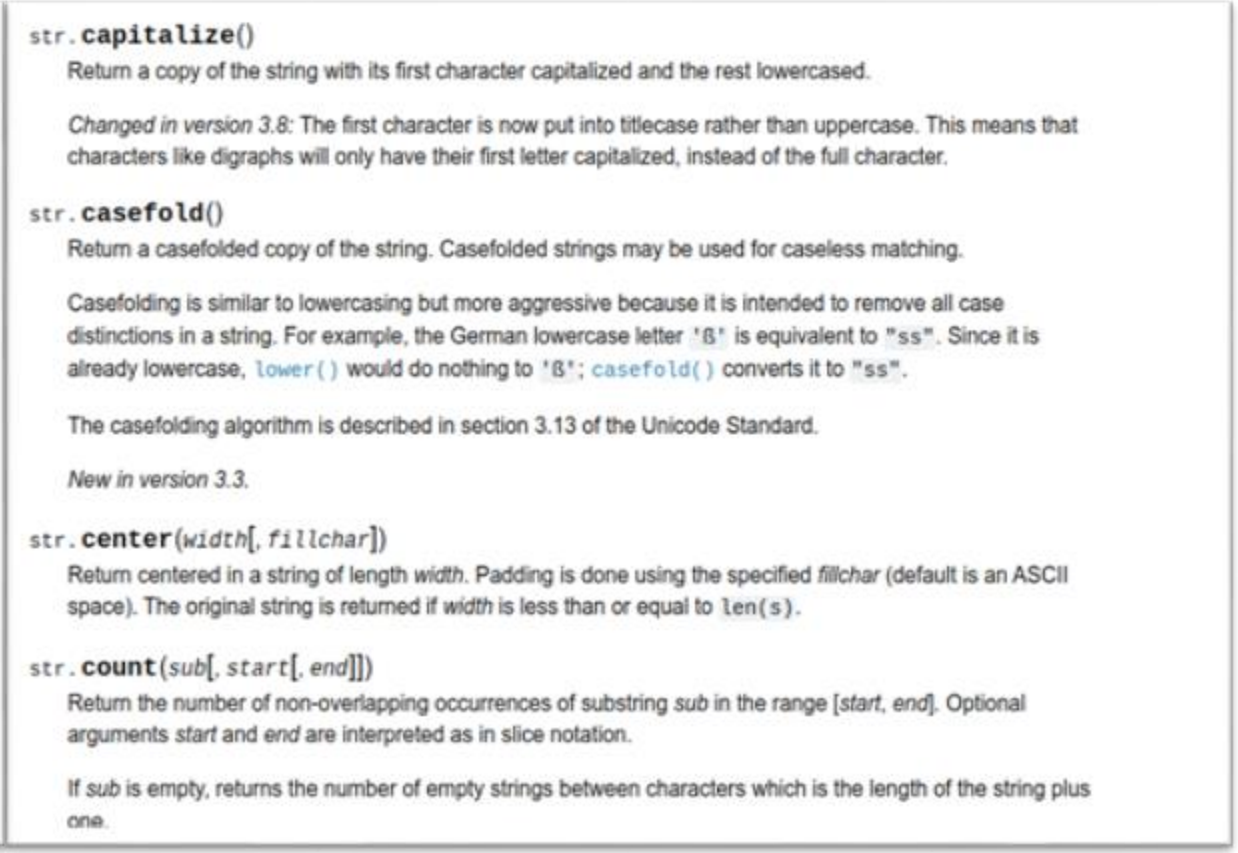
See `str.removesuffix()` for a method that will remove a single suffix string rather than all of a set of characters. For example:

```
>>> 'Monty Python'.rstrip(' Python')
'M'
>>> 'Monty Python'.removesuffix(' Python')
'Monty'
```

str.**split**(*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 (thus, the list will have at most `maxsplit+1` elements). If *maxsplit* is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`). The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2', '3']`). Splitting an empty string with a specified separator returns `['']`.



str.**capitalize**()

Return a copy of the string with its first character capitalized and the rest lowercased.

Changed in version 3.8: The first character is now put into titlecase rather than uppercase. This means that characters like digraphs will only have their first letter capitalized, instead of the full character.

str.**casefold**()

Return a casefolded copy of the string. Casefolded strings may be used for caseless matching.

Casefolding is similar to lowercasing but more aggressive because it is intended to remove all case distinctions in a string. For example, the German lowercase letter `'ß'` is equivalent to `"ss"`. Since it is already lowercase, `lower()` would do nothing to `'ß'`; `casefold()` converts it to `"ss"`.

The casefolding algorithm is described in section 3.13 of the Unicode Standard.

New in version 3.3.

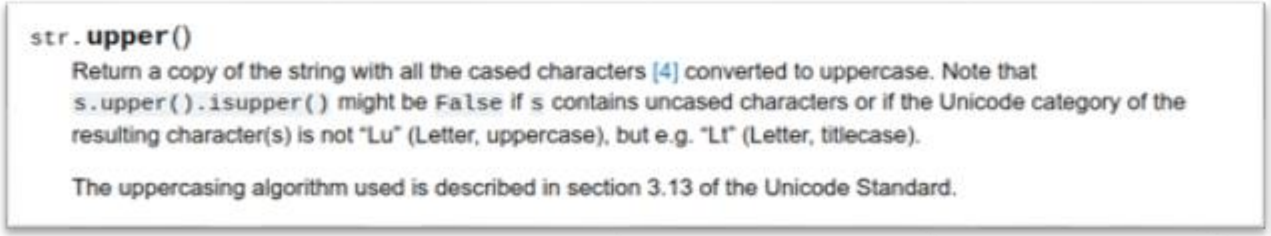
str.**center**(*width*[, *fillchar*])

Return centered 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.**count**(*sub*[, *start*[, *end*]])

Return the number of non-overlapping occurrences of substring *sub* in the range [*start*, *end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

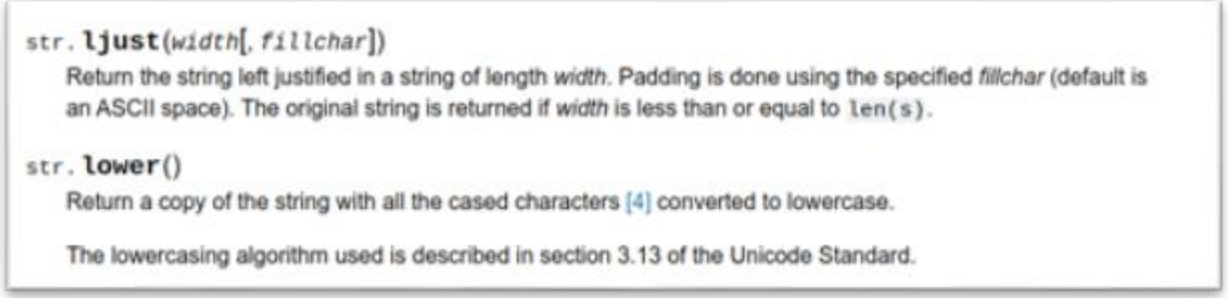
If *sub* is empty, returns the number of empty strings between characters which is the length of the string plus one.



str.**upper**()

Return a copy of the string with all the cased characters [4] converted to uppercase. Note that `s.upper().isupper()` might be `False` if `s` contains uncased characters or if the Unicode category of the resulting character(s) is not "Lu" (Letter, uppercase), but e.g. "Lt" (Letter, titlecase).

The uppercasing algorithm used is described in section 3.13 of the Unicode Standard.



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

Return the string left 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.**lower**()

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

The lowercasing algorithm used is described in section 3.13 of the Unicode Standard.

String Library

- str.capitalize()
- str.center(width[, fillchar])
- str.endswith(suffix[, start[, end]])
- str.find(sub[, start[, end]])
- str.lstrip([chars])
- str.replace(old, new[, count])
- str.lower()
- str.rstrip([chars])
- str.strip([chars])
- str.upper()

Square brackets '[]' mean optional parameters that may be used will calling the function.

Searching for a Substring

- We use the **find()** function to search for a substring within another string
- **find()** finds the first occurrence of the substring
- If the substring is not found, find() returns -1
- Remember that string position starts at zero

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

fruit.find('')?

Search and Replace

- The **replace()** function is like a “search and replace” operation in a word processor
- It replaces all occurrences of the search string with the replacement string

```
>>> greet = 'Hello Bob'
>>> nstr =greet.replace('Bob','Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o','X')
>>> print(nstr)
HellX BXb
```

Recall that Python string are immutable.

Splitting Strings

- **split()** is used to split a string into substrings.
- After performing the **split()**, the result is stored in a list of strings.
- Each string in the list is a substring after the split.

```
>>> s='IIT GANDHINAGAR'
>>> s.split()
['IIT', 'GANDHINAGAR']
>>> s.split('')
...
ValueError: empty separator
>>> s.split(' ')
['IIT', 'GANDHINAGAR']
>>> s.split('G')
['IIT ', 'ANDHINA', 'AR']
>>> s.split('AR')
['IIT GANDHINAG', '']
```

Stripping in Strings

- Remove whitespace from left using **lstrip()**
- Remove whitespace from right using **rstrip()**
- Remove whitespace from both ends using **strip()**

```
‘ ’ – Space
‘\t’ – Horizontal tab
‘\v’ – Vertical tab
‘\n’ – Newline
‘\r’ – Carriage return
‘\f’ – Feed
```

```
>>> s='   \t\nabc123 qertyw!    '
>>> s.lstrip()
'abc123 qertyw!    '
>>> s.rstrip()
'   \t\nabc123 qertyw!'
>>> s.strip()
'abc123 qertyw!'
```

Splitting Strings

```
>>> s='IIT GANDHINAGAR'
>>> s.split('G')
['IIT ', 'ANDHINA', 'AR']
>>> s.rsplit('G')
['IIT ', 'ANDHINA', 'AR']
>>> s.lsplit('G')
Traceback (most recent call last): File "", line 1, in
  AttributeError: 'str' object has no attribute 'lsplit'. Did you
  mean: 'rsplit'?
```

**Unlike `lstrip()`,
there is no `lsplit()`**

Lists

Collection: multiple items together

https://www.clarehall.cam.ac.uk/bellcollection/

What is Not a “Collection”?

- Most of our variables have one value in them - when we put a new value in the variable, the old value is overwritten

```
$ python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on
  linux Type "help", "copyright", "credits" or "license" for more
  information.
>>> x=2
>>> x=4
>>> print(x)
4
```

List as a Collection

- A collection allows us to put many values in a single “variable”
- A collection is nice because we can carry all many values (even of different types) around in one convenient package.

```
>>> x = 2
>>> x = 4
>>> print(x)
4
>>> x = [1, 2.35, True, 'A', 5]
print(x)
>>> [1, 2.35, True, 'A', 5]
```

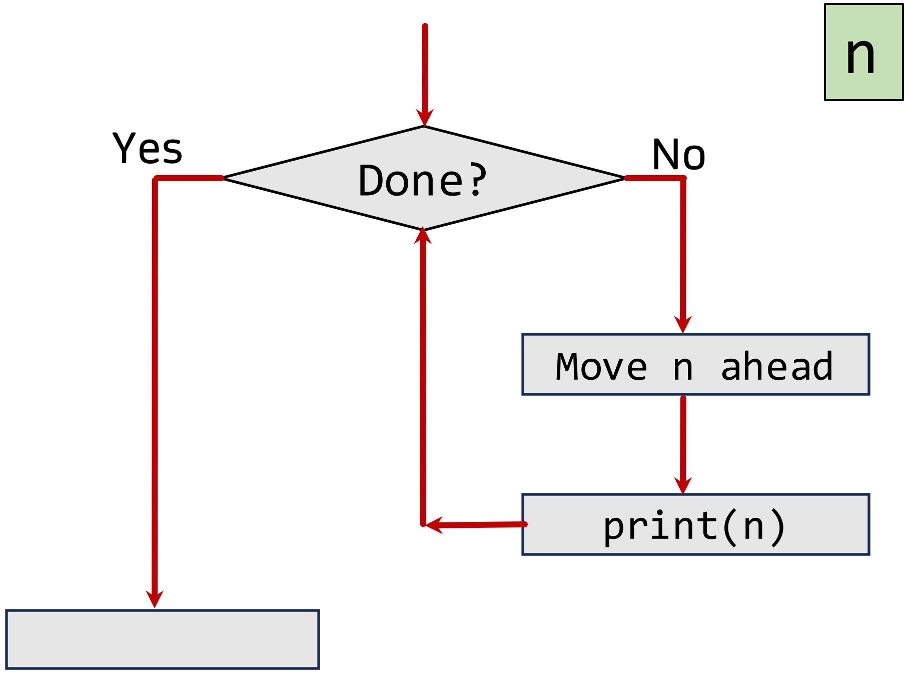
The indexing is exactly like strings. It can even go negative!

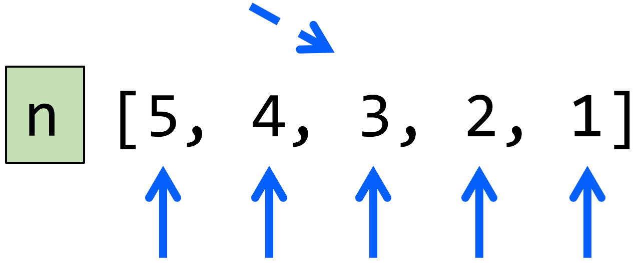
```
>>> print(x[0])

>>> print(x[-1])
5
x=[[],[]]
>>> print(len(x))
2
```

Why 2? Should it be zero?

Lists and Definite Loops

```
for n in [5, 4, 3, 2, 1]:
    print(n)
```





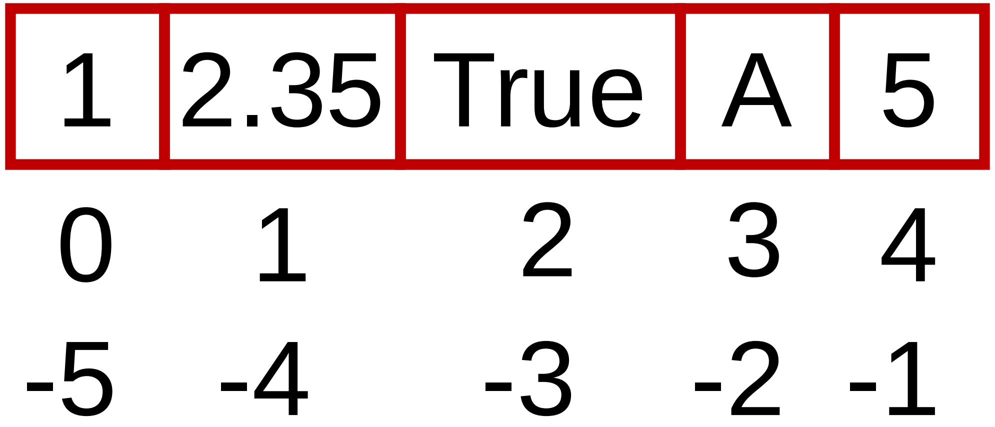
Output

```
5
4
3
2
1
```

Definite loops (for loops) have explicit iteration variables that change each time through a loop. These iteration variables move through the collection of items in order.

List indexing and slicing

- Just like strings, we can get at any single element in a list using an index/slice specified in square brackets



1	2.35	True	A	5
0	1	2	3	4
-5	-4	-3	-2	-1

```
>>> x = [1, 2.35, True, 'A', 5]
>>> print(x[-4])
2.35
>>> print(x[1:4])
[2.35, True, 'A']
>>> print(x[::-1])
[5, 'A', True, 2.35, 1]
>>> print(x[len(x)-1::-1])
[5, 'A', True, 2.35, 1]
```

Lists are Mutable

- Strings are “immutable” - we cannot change the contents of a string - we must make a new string to make any change
- Lists are “mutable” - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Contributors 2024 - Yogesh K. Meena and Shouvick Mondal, IIT Gandhinagar