CERTIFIED ASSOCIATE IN PYTHON PROGRAMMING

BY: IMRAN

The capitalize() method

- The capitalize() method does exactly what it says - **it creates a new string filled with characters taken from the source string**, but it tries to modify them in the following way:
  - **if the first character inside the string is a letter** (note: the first character is an element with an index equal to `0`, not just the first visible character), **it will be converted to upper-case**;
  - **all remaining letters from the string will be converted to lower-case**.
- Don't forget that:
  - the original string (from which the method is invoked) is not changed in any way (a string's immutability must be obeyed without reservation)
  - the modified (capitalized in this case) string is returned as a result - if you don't use it in any way (assign it to a variable, or pass it to a function/method) it will disappear without a trace.

The capi talize() method

- # Demonstrating the capitalize() method:
- print('aBcD'.capitalize())

The center() method

- The one-parameter variant of the center() method makes a copy of the original string, trying to center it inside a field of a specified width.
- The centering is actually done by **adding some spaces before and after the string**.
- # Demonstrating the center() method:
- print('[' + 'alpha'.center(10) + ']')
- print('[' + 'gamma'.center(20, '*') + ']')

The endswith() method

- he endswith() method **checks if the given string ends with the specified argument and returns True or False**, depending on the check result.
- Note: the substring must adhere to the string's last character - it cannot just be located somewhere near the end of the string.
-

```
t = "zeta"
print(t.endswith("a"))
print(t.endswith("A"))
print(t.endswith("et"))
print(t.endswith("eta"))
```

The find() method

- The find() method is similar to index(), which you already know - **it looks for a substring and returns the index of first occurrence of this substring**, but:
- it's safer - it **doesn't generate an error for an argument containing a non-existent substring** (it returns -1 then)
- it **works with strings only** - don't try to apply it to any other sequence.
- # Demonstrating the find() method:
- print("Eta".find("ta"))
- print("Eta".find("mma"))

The find () method

- t = 'theta'
- print(t.find('eta'))
- print(t.find('et'))
- print(t.find('the'))
- print(t.find('ha’))

- print('kappa'.find('a', 2))

The find () metho d

- t = 'theta'
- print(t.find('eta'))
- print(t.find('et'))
- print(t.find('the'))
- print(t.find('ha’))

- print('kappa'.find('a', 2))

The isalnum() method

- The parameterless method named isalnum() **checks if the string contains only digits or alphabetical characters (letters), and returns True or False** according to the result.
- Look at the example in the editor and run it.
- Note: any string element that is not a digit or a letter causes the method to return False. An empty string does, too.
- # Demonstrating the isalnum() method:
- print('lambda30'.isalnum())
- print('lambda'.isalnum())
- print('30'.isalnum())
- print('@'.isalnum())
- print('lambda_30'.isalnum())
- print(''.isalnum())

The isalpha() method

- The isalpha() method is more specialized - it's interested in **letters only**.

- # Example 1: Demonstrating the isapha() method:
- print("Moooo".isalpha())
- print('Mu40'.isalpha())

The isdigit() method

- In turn, the isdigit() method looks at **digits only** - anything else produces False as the result.
- # Example 2: Demonstrating the isdigit() method:
- print('2018'.isdigit())
- print("Year2019".isdigit())

Methods

- **The `islower()` method**
- The islower() method is a fussy variant of isalpha() - it accepts **lower-case letters only**.

- **The `isspace()` method**
- The isspace() method **identifies whitespaces only** - it disregards any other character (the result is False then).

- **The `isupper()` method**
- The isupper() method is the upper-case version of islower() - it concentrates on **upper-case letters only**.

Methods

- # Example 1: Demonstrating the islower() method:

- print("Moooo".islower())
- print('moooo'.islower())

- # Example 2: Demonstrating the isspace() method:
- print(' \n '.isspace())
- print(" ".isspace())
- print("mooo mooo mooo".isspace())

- # Example 3: Demonstrating the isupper() method:
- print("Moooo".isupper())
- print('moooo'.isupper())
- print('MOOOO'.isupper())

The join() method

- The join() method is rather complicated, so let us guide you step by step thorough it:
  - as its name suggests, the method **performs a join** - it expects one argument as a list; it must be assured that all the list's elements are strings - the method will raise a `TypeError` exception otherwise;
  - all the list's elements will be **joined into one string** but...
  - ...the string from which the method has been invoked is **used as a separator**, put among the strings;
  - the newly created string is returned as a result.

- # Demonstrating the join() method:

- print(",".join(["omicron", "pi", "rho"]))

The lower() method

- The lower() method **makes a copy of a source string, replaces all upper-case letters with their lower-case counterparts**, and returns the string as the result. Again, the source string remains untouched.
- If the string doesn't contain any upper-case characters, the method returns the original string.
- Note: The lower() method doesn't take any parameters.

- # Demonstrating the lower() method:

- print("SiGmA=60".lower())

The lstrip() method

- The parameterless lstrip() method **returns a newly created string formed from the original one by removing all leading whitespaces**.
-

```
# Demonstrating the lstrip() method:
```

- `print("[" + " tau ".lstrip() + "]")`

- The **one-parameter** lstrip() method does the same as its parameterless version, but **removes all characters enlisted in its argument** (a string), not just whitespaces:

- `print("www.cisco.com".lstrip("w."))`

The replace() method

- The **two-parameter** replace() method **returns a copy of the original string in which all occurrences of the first argument have been replaced by the second argument**.
- # Demonstrating the replace() method:
- print("www.netacad.com".replace("netacad.com", "pythoninstitute.org"))
- print("This is it!".replace("is", "are"))
- print("Apple juice".replace("juice", ""))

The repl ace() me thod

- If the second argument is an empty string, **replacing is actually removing** the first argument's string. What kind of magic happens if the first argument is an empty string?
- The **three-parameter** replace() variant uses the third argument (a number) to **limit the number of replacements**.
- print("This is it!".replace("is", "are", 1))
- print("This is it!".replace("is", "are", 2))

Module 2: Strings, Lists, and Exceptions
Part 3: Strings Methods

Module 2: Strings, Lists, and Exceptions
Part 3: Strings Methods

Module 2: Strings, Lists, and Exceptions
Part 3: Strings Methods

ImranNust

imran_muet

muhammad-imran-b7865495

The rfind() method

- The one-, two-, and three-parameter methods named rfind() do nearly the same things as their counterparts (the ones devoid of the *r* prefix), but **start their searches from the end of the string**, not the beginning (hence the prefix *r*, for *right*).

- `# Demonstrating the rfind() method:`
- `print("tau tau tau".rfind("ta"))`
- `print("tau tau tau".rfind("ta", 9))`
- `print("tau tau tau".rfind("ta", 3, 9))`

The rstrip() method

- Two variants of the rstrip() method do nearly the same as lstrips, but **affect the opposite side of the string**.
- # Demonstrating the rstrip() method:
- print("[" + " upsilon ".rstrip() + "]")
- print("cisco.com".rstrip(".com"))

The split() method

- he split() method does what it says - it **splits the string and builds a list of all detected substrings**.
- The method **assumes that the substrings are delimited by whitespaces** - the spaces don't take part in the operation, and aren't copied into the resulting list.
- If the string is empty, the resulting list is empty too.
- # Demonstrating the split() method:

- print("phi   chi\npsi".split())

The startswith() method

- The startswith() method is a mirror reflection of endswith() - it **checks if a given string starts with the specified substring**.
- # Demonstrating the startswith() method:
- print("omega".startswith("meg"))
- print("omega".startswith("om"))

The strip() method

- The strip() method combines the effects caused by rstrip() and lstrip() - it **makes a new string lacking all the leading and trailing whitespaces**.

- # Demonstrating the strip() method:

- print("[" + "   aleph   ".strip() + "]")

The swapcase() method

- The swapcase() method **makes a new string by swapping the case of all letters within the source string**: lower-case characters become upper-case, and vice versa.

- # Demonstrating the swapcase() method:

- print("I know that I know nothing.".swapcase())

The title() method

- The title() method performs a somewhat similar function - it **changes every word's first letter to upper-case, turning all other ones to lower-case**.
- # Demonstrating the title() method:
- print("I know that I know nothing. Part 1.".title())

Module 2: Strings, Lists, and Exceptions
Part 3: Strings Methods

The upper() method

- Last but not least, the upper() method **makes a copy of the source string, replaces all lower-case letters with their upper-case counterparts**, and returns the string as the result.
- # Demonstrating the upper() method:
- print("I know that I know nothing. Part 2.".upper())