Lecture 11: Strings

Swakkhar Shatabda

B.Sc. in Data Science
Department of Computer Science and Engineering
United International University

January 17, 2024

Formatting Strings

```
print(f'{17.489:.2f}')
```

- Python supports precision only for floating-point and Decimal values.
- Formatting is type dependent—if you try to use .2f to format a string like 'hello', a ValueError occurs.
- presentation type : f,d,c,s

```
print(f'{10:d}')
print(f'{65:c} {97:c}')
print(f'{"hello":s} {7}')
```

Decimal and Floting point values

- For extremely large and small values of these types, Exponential (scientific) notation can be used to format the values more compactly.

```
from decimal import Decimal
print(f'{Decimal("10000000000000000000000000.0"):.3f}')
print(f'{Decimal("10000000000000000000000000.0"):.3e}')
```

Field Widths

- By default, Python right-aligns numbers and left-aligns other values such as strings—we enclose the results below in brackets ([]) so you can see how the values align in the field.

```
print(f'[{27:10d}]')
print(f'[{3.5:10f}]')
print(f'[{"hello":10}]')
```

Alignment

- you can specify left and right alignment with $<$ and $>$

```
print(f'[{27:<15d}]')
print(f'[{3.5:<15f}]')
print(f'[{"hello":>15}]')
```

- Centering attempts to spread the remaining unoccupied character positions equally to the left and right of the formatted value. Python places the extra space to the right if an odd number of character positions remain.

```
print(f'[{27:^7d}]')
print(f'[{3.5:^7.1f}]')
print(f'[{"hello":^7}]')
```

Concatenating and Repeating Strings

- the + operator to concatenate strings and the * operator to repeat strings.

```
s1 = 'happy'
s2 = 'birthday'
s1 += ' ' + s2
print(s1)
symbol = '>'
symbol *= 5
print(symbol)
```

Striping

- strip to remove the leading and trailing whitespace from a string.
- lstrip and rstrip for leading and trailing spaces.

```
sentence = '\t  \n  This is a test string. \t\t \n'
print(sentence.strip())
```

Changing Character Case

- Method capitalize copies the original string and returns a new string with only the first letter capitalized (this is sometimes called sentence capitalization)

```
print('happy birthday'.capitalize())
```

- Method title copies the original string and returns a new string with only the first char- acter of each word capitalized (this is sometimes called book-title capitalization)

```
print('strings: a deeper look'.title())
```

- There are also lower and upper.

String Comparison

- ord function gives you the ASCII values.

```
print('Orange' == 'orange')
print('Orange' != 'orange')
print('Orange' < 'orange')
print('Orange' <= 'orange')
print('Orange' > 'orange')
print('Orange' >= 'orange')
```

Searching for Substrings

- String method count returns the number of times its argument occurs in the string on which the method is called.

```
sentence = 'to be or not to be that is the question'
print(sentence.count('to'))
```

- If you specify as the second argument a start_index, count searches only the slice string from start_index through end of the string.

```
sentence = 'to be or not to be that is the question'
print(sentence.count('to',12))
```

- If you specify as the second and third arguments the start_index and end_index, count searches only the slice from start_index up to, but not including, end_index.

Locating a substring

- String method index searches for a substring within a string and returns the first index at which the substring is found; otherwise, a ValueError occurs.

```
sentence = 'to be or not to be that is the question'
print(sentence.index('be'))
```

- String method rindex performs the same operation as index, but searches from the end of the string and returns the last index at which the substring is found; otherwise, a Value- Error occurs.

Locating substrings

- If you need to know only whether a string contains a substring, use operator in or not in.
- String methods startswith and endswith return True if the string starts with or ends with a specified substring.

```
sentence = 'to be or not to be that is the question'
print('THAT' in sentence)
print('that' in sentence)
print('THAT' not in sentence)
print(sentence.startswith('to'))
print(sentence.endswith('question'))
```

Replace tokens

- A common text manipulation is to locate a substring and replace its value.
- Method replace takes two substrings.
- It searches a string for the substring in its first argument and replaces each occurrence with the substring in its second argument.
- The method returns a new string containing the results.

```
values = '1\t2\t3\t4\t5'
print(values.replace('\t', ','))
```

Splitting Strings

- string method split with no arguments tokenizes a string by breaking it into substrings at each whitespace character, then returns a list of tokens.
- To tokenize a string at a custom delimiter (such as each comma-and-space pair), specify the delimiter string (such as, ', ') that split uses to tokenize the string.
- If you provide an integer as the second argument, it specifies the maximum number of splits. The last token is the remainder of the string after the maximum number of splits

```
letters = 'A, B, C, D'
print(letters.split(', '))
```

Joining Strings

- String method join concatenates the strings in its argument, which must be an iterable containing only string values; otherwise, a TypeError occurs.
- The separator between the concatenated items is the string on which you call join.
- The following code creates strings containing comma-separated lists of values

```
letters_list = ['A', 'B', 'C', 'D']
print(','.join(letters_list))
print(','.join([str(i) for i in range(10)]))
```

Partition

- String method partition splits a string into a tuple of three strings based on the method's separator argument. The three strings are
  - the part of the original string before the separator
  - the separator itself, and
  - the part of the string after the separator.

```
print('Amanda: 89, 97, 92'.partition(': '))
```

- To search for the separator from the end of the string instead, use method rpartition to split.

```
url = 'http://www.deitel.com/books/PyCDS/
table_of_contents.html'
rest_of_url, separator, document = url.rpartition('/')
```

Testing Characters

String Method	Description
`isalnum()`	Returns `True` if the string contains only *alphanumeric* characters (i.e., digits and letters).
`isalpha()`	Returns `True` if the string contains only *alphabetic* characters (i.e., letters).
`isdecimal()`	Returns `True` if the string contains only *decimal integer* characters (that is, base 10 integers) and does not contain a + or - sign.
`isdigit()`	Returns `True` if the string contains only digits (e.g., `'0'`, `'1'`, `'2'`).
`isidentifier()`	Returns `True` if the string represents a valid *identifier*.
`islower()`	Returns `True` if all alphabetic characters in the string are *lowercase* characters (e.g., `'a'`, `'b'`, `'c'`).
`isnumeric()`	Returns `True` if the characters in the string represent a *numeric value* without a + or - sign and without a decimal point.
`isspace()`	Returns `True` if the string contains only *whitespace* characters.
`istitle()`	Returns `True` if the first character of each word in the string is the only *uppercase* character in the word.
`isupper()`	Returns `True` if all alphabetic characters in the string are *uppercase* characters (e.g., `'A'`, `'B'`, `'C'`).