More about Strings

CAC 180

Amber Wagner

Strings

- We already know what strings are
- We know how to manipulate strings in a few ways…
- Can you name some ways to manipulate strings?

Slice Notation

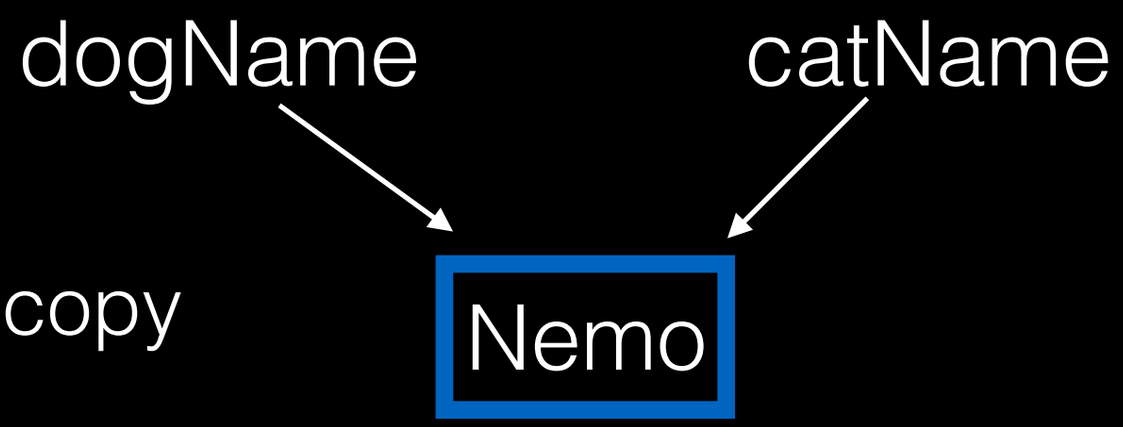
- newString = originalString[start:end]
- The end is one spot after the location of the desired data
- Negative positions can be used for start and end positions
- lists and tuples also support slice notation

Slice Notation

- The slicing creates a new object in memory
- If we set two strings equal to one another, that means that there are two references to the same object in memory

dogName = ‘Nemo’

catName = dogName



- If we use slicing, it makes a copy

catName = dogName[:]

Slice Notation

website
cats.com

domain
com

- website = ‘cats.com’
- domain = website[-3:]
- website = ‘cats.org’

website
cats.org

Slice Notation

Table 7.1.1: Common slicing operations.

A list of common slicing operations a programmer might use.
Assume the value of my_str is 'http://en.wikipedia.org/wiki/Nasa/'

Syntax	Result	Description
my_str[10:19]	wikipedia	Gets the characters in positions 10-18.
my_str[10:-5]	wikipedia.org/wiki/	Gets the characters in positions 10-28.
my_str[8:]	n.wikipedia.org/wiki/Nasa/	All characters from position 8 until the end of the string.
my_str[:23]	http://en.wikipedia.org	Every character up to position 23, but not including my_str[23].
my_str[:-1]	http://en.wikipedia.org/wiki/Nasa	All but the last character.
my_str[:]	http://en.wikipedia.org/wiki/Nasa/	A new copy of the my_str object.

Stride

- Let’s say we only want every other letter in a sentence

sentence = ‘Rain rain go away’

everyOther = sentence[::2]

- The third position represents the stride - the amount by which to increment the index as the string is being traversed

Stride and String Reversal

- Stride allows us to do a really need trick to reverse a string…

sentence = ‘Rain rain go away’

reverse = sentence[::-1]

Advanced String Formatting

- To format output in a specific way (e.g., leading zeroes, number of significant digits)
- Conversion specifier: placeholder for a value in a string literal
- Ex: print(‘age is %d’ % (user_age))

Conversion Types

Table 3.7.1: Common conversion specifiers.

Conversion specifier(s)	Notes	Example	Output
%d	Substitute as integer.	`print('%d' % 10)`	`10`
%f	Substitute as floating-point decimal	`print('%f' % 15.2)`	`15.200000`
%s	Substitute as string.	`print('%s' % 'ABC')`	`ABC`
%x, %X	Substitute as hexadecimal in lowercase (%x) or uppercase (%X).	`print('%x' % 31)`	`1f`
%e, %E	Substitute as floating-point exponential format in lowercase (%e) or uppercase (%E).	`print('%E' % 15.2)`	`1.520000E+01`

Conversion Specifiers

'Tokyo' 35.600000 140.800000

```
'%s: %f North, %f East' % ('Tokyo', 35.6, 140.8)
```

'Tokyo: 35.600000 North, 140.800000 East'

'Tokyo' 35 140

```
'%s: %d North, %d East' % ('Tokyo', 35.6, 140.8)
```

'Tokyo: 35 North, 140 East'

- Just like in function parameters, if there are multiple conversion specifiers, the number of elements in the tuple, must match
- The values in the tuple should be placed in the order in which the specifiers appear

Aligning Data

- If you want to print out information so that it appears to be in columns, you would want to ensure each column took up the same amount of space

Figure 7.2.1: String formatting example: Add leading 0s by setting the minimum field width and 0 conversion flag.

```
student_id = int(input('Enter student ID: '))

print('The user entered %d' % student_id)
print('Full 8-character student ID: %08d' % student_id)
```

```
Enter student ID: 1234
The user entered 1234
Full 8-character student ID: 00001234
```

Helpful String Methods

- To use these methods, type stringName.methodName
  - replace(old, new)
  - replace(old, new, count)
  - find(x)
  - find(x, start)
  - find(x, start, end)
  - rfind(x)
  - count(x)

More Helpful String Methods

- To use these methods, type stringName.methodName
  - isaInum()
  - isdigit()
  - islower()
  - isupper()
  - isspace()
  - startswith(x)
  - endswith(x)

More Helpful String Methods

- To use these methods, type stringName.methodName
  - capitalize()
  - lower()
  - upper()
  - strip()
  - title()

String Comparison

myName = ‘Wagner’

myLastName = ‘Wagner’

- What is the result of the following comparisons?

myName == myLastName

myName is myLastName

String Comparison

- When comparing if one string is less than or greater than another string, it does a lexicographical comparison
- Essentially, which one comes first in the dictionary?

Splitting Strings

- When we use the split() method, we can split on a string into substrings or tokens

- When we did the Pirate Translator, you probably used split() without any parameters.
  - By default, the separator is a space
  - You can split on any character though

Figure 7.4.1: String split example.

```
url = input('Enter URL:\n')

tokens = url.split('/')  # Uses '/' separator
print(tokens)
```

```
Enter URL: http://en.wikipedia.org/wiki/Lucille_ball
['http:', '', 'en.wikipedia.org', 'wiki', 'Lucille_ball']
...
Enter URL: en.wikipedia.org/wiki/ethernet/
['en.wikipedia.org', 'wiki', 'ethernet', '']
```

Joining Strings

- The join() method does the opposite of split()
- It takes a list of strings (tokens) and joins each of them together with a specified character

Figure 7.4.2: String join example: Comparing join vs. loops.

The following programs are equivalent, however join() is a simpler approach, using less code and being easier to read.

```
phrases = ['To be, ', 'or not to be.\n', 'That is the question.']

sentence = ''
for phrase in phrases:
    sentence += phrase
print(sentence)
```

```
To be, or not to be.
That is the question.
```

```
phrases = ['To be, ', 'or not to be.\n', 'That is the question.']

sentence = ''.join(phrases)
print(sentence)
```

```
To be, or not to be.
That is the question.
```

Sorting a String

- There may be a reason why you would want to sort the letters in a string
- Strings are immutable so we can't do this in place, but there are some tricks we can use:
  - sorted(*someString*) will sort the letters, but it returns a list
  - We can then use join() to put the letters together in a string

Sorting a String

- Example:
- name = ‘Amber Wagner’
- sortedName = ‘’.join(sorted(name))

Format Method

- Originally designed to replace the % formatting notation - to improve readability

1. *Positional*: An integer that describes the position of the value.

```
'The {1} in the {0}'.format('hat','cat')
```

```
The cat in the hat
```

2. *Inferred positional*: Empty {} assumes ordering of replacement fields is {0}, {1}, {2}, etc.

```
'The {} in the {}'.format('cat', 'hat')
```

```
The cat in the hat
```

3. *Named*: A name matching a keyword argument.

```
'The {animal} in the {headwear}'.format(animal='cat', headwear='hat')
```

```
The cat in the hat
```

Practice

- Using what you just learned, solve the following problems:

1. Check if a word is a palindrome (same forward and backward like racecar)

2. Check if two strings are anagrams.

3. Print out, “The solution is 16.8” where 16.8 is in exponential format.

Next Class

- Review Chapter 8

- Don’t forget to be doing the practice problems in the book (participation points)