

Strings

Strings

Three ways to define string literals:

- ▶ with single quotes: 'Ni!'
- ▶ double quotes: "Ni!"
- ▶ Or with triples of either single or double quotes, which creates a multi-line string:

```
1 >>> """I do HTML for them all,  
2 ... even made a home page for my dog."""  
3 'I do HTML for them all,\neven made a home page for my  
dog.'
```

Strings

Note that the REPL echoes the value with a `\n` to represent the newline character. Use the `print` function to get your intended output:

```
1 >>> nerdy = """I do HTML for them all,  
2 ... even made a home page for my dog."""  
3 >>> nerdy  
4 'I do HTML for them all,\nneven made a home page for my dog.'  
5 >>> print(nerdy)  
6 I do HTML for them all,  
7 even made a home page for my dog.
```

That's pretty nerdy.

Strings

Choice of quote character is usually a matter of taste, but the choice can sometimes buy convenience. If your string contains a quote character you can either escape it:

```
1 >>> journey = 'Don\'t stop believing.'
```

or use the other quote character:

```
1 >>> journey = "Don't stop believing."
```

► How does Python represent the value of the variable `journey` ?

String Operations

Because strings are sequences we can get a string's length with

`len()`:

```
1 >>> i = "team"
2 >>> len(i)
3 4
```

and access characters in the string by index (offset from beginning – first index is 0) using `[]`:

```
1 >>> i[1]
2 'e'
```

Note that the result of an index access is a string:

```
1 >>> type(i[1])
2 <class 'str'>
3 >>> i[3] + i[1]
4 'me'
5 >>> i[-1] + i[1] # Note that a negative index goes from the
6                  end
7 'me'
```

► What is the index of the first character of a string?

► What is the index of the last character of a string?

String Slicing

`[:end]` gets the first characters up to but not including `end`

```
1 >>> al_gore = "manbearpig"
2 >>> al_gore[:3]
3 'man'
```

`[begin:end]` gets the characters from `begin` up to but not including `end`

```
1 >>> al_gore[3:7]
2 'bear'
```

`[begin:]` gets the characters from `begin` to the end of the string

```
1 >>> al_gore[7:]
2 'pig'
3 >>>
```

- What is the relationship between the ending index of a slice and the beginning index of a slice beginning right after the first slice?

String Methods

`str` is a class (you'll learn about classes later) with many methods (a method is a function that is part of an object). Invoke a method on a string using the dot operator.

`str.find(substr)` returns the index of the first occurrence of `substr` in `str`

```
1 >>> 'foobar'.find('o')
2 1
```

- ▶ Write a string slice expression that returns the username from an email address, e.g., for 'bob@aol.com' it returns 'bob'.
- ▶ Write a string slice expression that returns the host name from an email address, e.g., for 'bob@aol.com' it returns 'aol.com'.

String Interpolation with %

The old-style (2.X) string format operator, %, takes a string with format specifiers on the left, and a single value or tuple of values on the right, and substitutes the values into the string according to the conversion rules in the format specifiers. For example:

```
1 >>> "%d %s %s %s %f" % (6, 'Easy', 'Pieces', 'of', 3.14)
2 '6 Easy Pieces of 3.140000'
```

Here are the conversion rules:

- ▶ %s string
- ▶ %d decimal integer
- ▶ %x hex integer
- ▶ %o octal integer
- ▶ %f decimal float
- ▶ %e exponential float
- ▶ %g decimal or exponential float
- ▶ %% a literal

String Formatting with %

Specify field widths with a number between % and conversion rule:

```
1 >>> sunbowl2012 = [('Georgia Tech', 21), ('USC', 7)]
2 >>> for team in sunbowl2012:
3 ...     print('%14s %2d' % team)
4 ...
5 Georgia Tech 21
6 USC          7
```

Fields right-aligned by default. Left-align with - in front of field width:

```
1 >>> for team in sunbowl2012:
2 ...     print('%-14s %2d' % team)
3 ...
4 Georgia Tech 21
5 USC          7
```

Specify n significant digits for floats with a .n after the field width:

```
1 >>> '%5.2f' % math.pi
2 ' 3.14'
```

Notice that the field width includes the decimal point and output is left-padded with spaces

String Interpolation with `str.format()`

Python 3.0 - 3.5 interpolation was done with the string method

`format`:

```
1 >>> "{} {} {} {} {}".format(6, 'Easy', 'Pieces', 'of', 3.14)
2 '6 Easy Pieces of 3.14'
```

Old-style formats only resolve arguments by position. New-style formats can take values from any position by putting the position number in the `{}` (positions start with 0):

```
1 >>> "{4} {3} {2} {1} {0}".format(6, 'Easy', 'Pieces', 'of',
2   3.14)
   '3.14 of Pieces Easy 6'
```

Can also use named arguments, like functions:

```
1 >>> "{count} pieces of {kind} pie".format(kind='punkin',
2   count=3)
   '3 pieces of punkin pie'
```

Or dictionaries (note that there's one dict argument, number 0):

```
1 >>> "{0[count]} pieces of {0[kind]}
2   pie".format({'kind': 'punkin',
   count: 3})
```

String Formatting with `str.format()`

Conversion types appear after a colon:

```
1 >>> "{:d} {} {} {} {:f}".format(6, 'Easy', 'Pieces', 'of',  
2   3.14)  
'6 Easy Pieces of 3.140000'
```

Argument names can appear before the :, and field formatters appear between the : and the conversion specifier (note the < and > for left and right alignment):

```
1 >>> for team in sunbowl2012:  
2     ...     print('{:<14s} {:>2d}'.format(team[0], team[1]))  
3     ...  
4 Georgia Tech 21  
5 USC          7
```

You can also unpack the tuple to supply its elements as individual arguments to format (or any function) by prepending tuple with *:

```
1 >>> for team in sunbowl2012:  
2     ...     print('{:<14s} {:>2d}'.format(*team))  
3     ...  
4 Georgia Tech 21  
5 USC          7
```

f-Strings

Python 3.6 introduced a much more convenient inline string interpolator. Prepend `f` to the opening quote, enclose arbitrary Python expressions in curly braces (`{}`), and put formatters similar to `str.format()` after colons.

```
1 >>> for team, score in sunbowl2012:           # Tuple-unpacking
      assignment
2     ...     print(f'{team:<14s} {score:>2d}')
3     ...
4 Georgia Tech      21
5 USC               7
```

Conclusion

Your turn:

- ▶ Try Exercise 1 listed in the schedule for today's lesson.