Module 6

Strings

Advanced String Expressions

An Interesting Problem

- Characters include punctuation ('Hello!')
- What if we want to put a quote in a string?
  - **Example**:

D	o	n	'	t

  - **Problem**: 'Don't' $\Rightarrow$

D	o	n

    $+$????
  - **Solution**: "Don't"
- But double quote does not always work
  - **Example**:

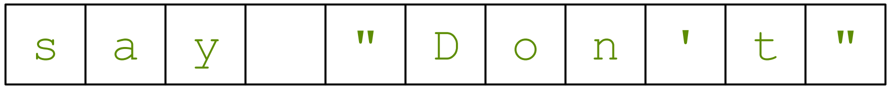
s	a	y		"	H	e	l	l	o	"

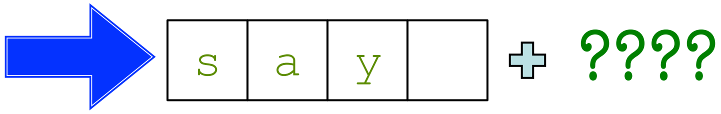
  - **Problem**: "say "Hello"" $\Rightarrow$

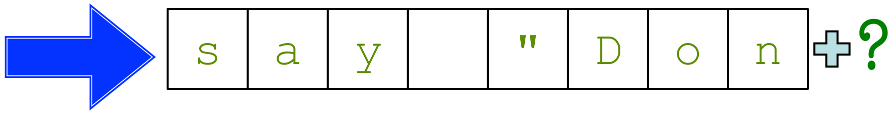
s	a	y	

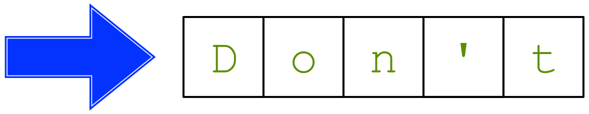
    $+$???
  - **Solution:** 'say "Hello"'

An Interesting Problem

- What if we combine the two?
  - 

  - **Problem**: "say "Don't""


  - **Problem**: 'say "Don't"'


  - **Solution**: ????
- Actual solution is escape characters
  - Way to tell python that a (quote) character is in box
  - Do this with a backslash: \
  - **Example:** 'Don\'t'


Other Escape Characters

- What if want to include the backslash?
  - **Example**: '\\' $\Rightarrow \boxed{\texttt{'}}$, error
  - **Solution**: '\\\\' $\Rightarrow \boxed{\texttt{\textbackslash}}$
  - First \\ is the *escape*, second is the *character*
  - Together they are an escape character
- There are many other examples
  - Often for formatting text
  - New lines, adding tabs
  - Visible with `print` functions

Char	Meaning
`\'`	single quote
`\"`	double quote
`\n`	new line
`\t`	tab
`\\`	backslash

Print and Escape Characters

```
>>> print('Hello\nWorld')
Hello
World
>>> print('Hello\tWorld')
Hello	World
>>> print('a\\b\\c')
a\b\c
>>> print('\\\\\\\\')
\\\\
```

print can help
you see the "boxes"

String Slicing

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[3:6]?

A: 'lo a'
B: 'lo'
C: 'lo ' **CORRECT**
D: 'o '
E: I do not know

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called “string slicing”

- s = 'a\\b\'c’

0	1	2	3	4
a	\	b	'	c

- Slicing shows “boxes”
  - s[1] is '\\'
  - s[3] is '\''
- These are one character!
  - len(s[1]) is 1, not 2
  - len(s[3]) is also 1
  - len(s) is 5, not 7

Other Important Ideas

Negative Indices

```
>>> s = 'Hello all'
>>> s[-1]
'l'
>>> s[-3]
'a'
>>> s[1:-1]
'ello al'
```

Variables as Indices

```
>>> s = 'Hello all'
>>> x = 2
>>> y = 7
>>> s[x:y]
'llo a'
>>> s[x+2:y]
'o a'
```

String Methods

Strings Have Few Functions

- Strings have very few built-in functions
  - We have already seen len, print, (and input)
  - Not much else without going to modules
- That is because strings use **methods** instead
  - Method calls act a lot like function calls
  - They are just written somewhat differently
- Why methods and not functions?
  - We will see why later in the course

Strings Have Few Functions

- Strings have very few built-in functions
  - We have already seen len, print, (and input)
  - Not much else without going to modules
- That is [illegible] tead
  - Meth[illegible]
  - They are just structured differently
- Why methods and not functions?
  - We will see why later in the course

Right now, only learning to **call** methods, not **define** them

Function Calls vs Method Calls

Function Call	Method Call
$\underbrace{\textbf{name}}_{\textbf{function name}}(\underbrace{x}_{\textbf{argument}})$	$\underbrace{\textbf{string}}_{\textbf{argument}}.\underbrace{\textbf{name}}_{\textbf{method name}}(\ )$

Right now, assume only **one** argument

Example: upper()

- upper(): Return an upper case **copy**

```
>>> s = 'Hello World'
>>> s.upper()
'HELLO WORLD'
>>> s[1:5].upper()    # Str before need not be a variable
'ELLO'
>>> 'abc'.upper()     # Str before could be a literal
'ABC'
```

- Notice that *only* argument is string in front

Alternative: Introcs

- The `introcs` module does have string functions
- In fact, it has a function form of upper

```
>>> import introcs
>>> s = 'Hello World'
>>> introcs.upper(s)
'HELLO WORLD'
```

- **Idea:** Alternative if you struggle with methods
  - But made for a very different type of course
  - In this course, we should learn methods

Advanced String Methods

String Methods

- In a previous video we saw method calls

$$\underbrace{\textbf{string}}_{\textbf{argument}}\textbf{.}\underbrace{\textbf{name}}_{\textbf{method name}}\textbf{( )}$$

- **Example:** `'Hello'.upper()`
- But it only has a single argument
  - Functions could have multiple arguments
  - Can methods have additional arguments too?

Additional Arguments

- Additional arguments go inside of parentheses

$$\underbrace{\textbf{string}}_{\textbf{argument}}\textbf{.}\underbrace{\textbf{name}}_{\textbf{method name}}(\underbrace{x,y}_{\textbf{additional arguments}},\ldots)$$

- But first argument (string) is always in front

Examples of String Methods

- s_1.index(s_2)
  - Returns position of the *first* instance of s_2 in s_1
- s_1.count(s_2)
  - Returns number of times s_2 appears inside of s_1
- s.strip()
  - Returns copy of s with no white-space at *ends*

```
>>> s = 'abracadabra'
>>> s.index('a')
0
>>> s.index('rac')
2
>>> s.count('a')
5
>>> s.count('x')
0
>>> '  a b '.strip()
'a b'
```

Examples of String Methods

- s_1.index(s_2)
  - Returns position of the *first* instance of s_2 in s_1
- s_1.count(s
  - Return
    s_2 appe
- s.strip()
  - Returns copy of s with no white-space at *ends*

See Lecture page for more

```
>>> s = 'abracadabra'
>>> s.index('a')
0
                 rac')

                  a')
5
>>> s.count('x')
0
>>> '  a b '.strip()
'a b'
```

Example: upper()

```
>>> s = 'Hello World’
>>> s.upper()
'HELLO WORLD’
>>> s[1:5].upper()
'ELLO’
>>> 'abc'.upper()
'ABC'
```

Replaces
introcs.upper()

Example: count

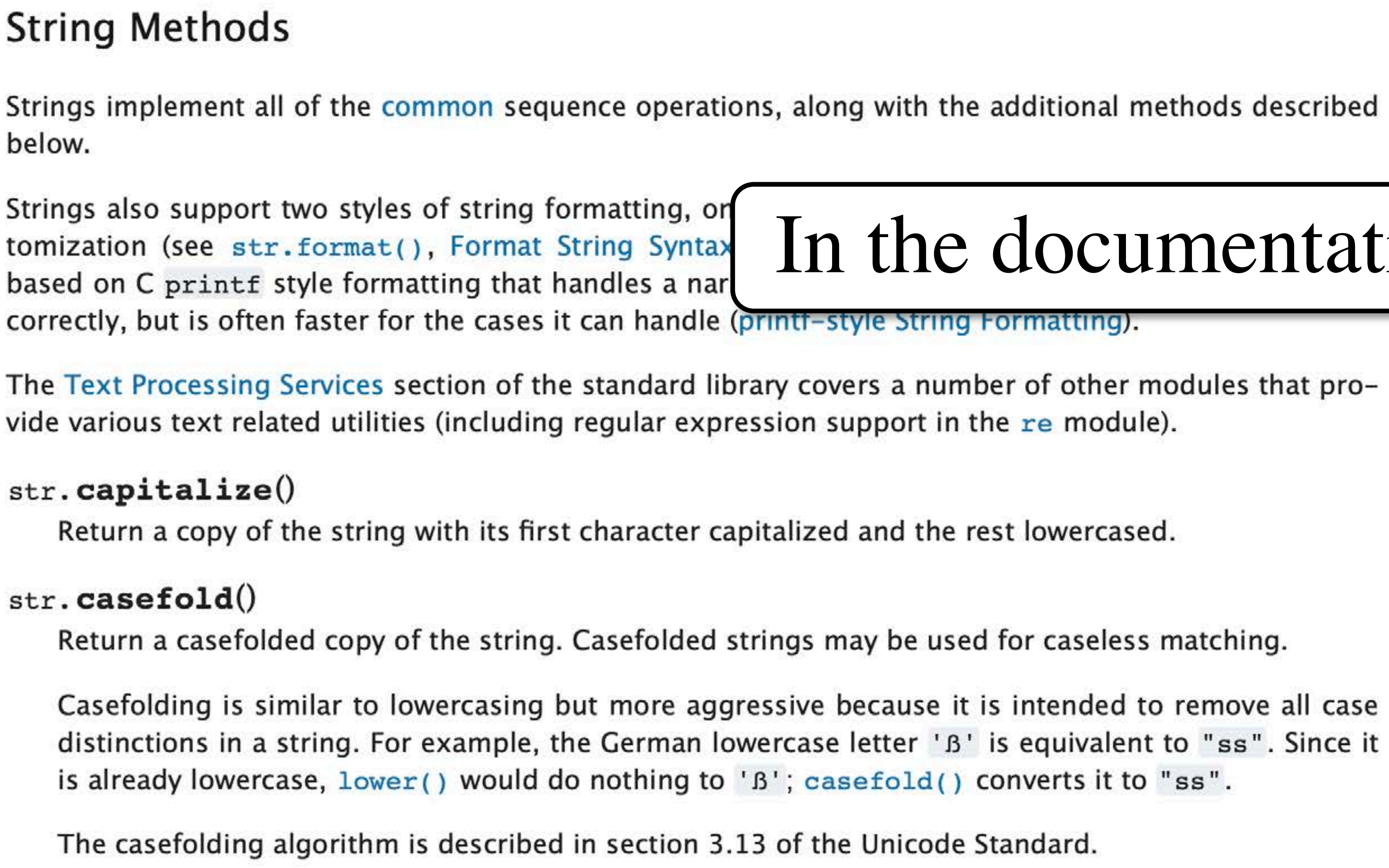
- **Format**: s_1.count(s_2)
  - Number of times s_2 appears inside of s_1
  - The string you search **for** is in parentheses!
- **Examples:**
  - `s = 'abbac'`
  - `s.count('a') == 2`
  - `s.count('c') == 1`
  - `s.count('x') == 0`
  - `s.count('ab') == 1`

Example: index

- **Format**: s_1.index(s_2)
  - Position of the **first** instance of s_2 in s_1
  - Same argument order as `count_str`
- **Examples**:
  - `s = 'abbac'`
  - `s.index('c') == 4`
  - `s.index('a') == 0`
  - `s.index('x') CRASHES`
  - `s.index('ab') == 0`

find is a
kinder variant

Where To Learn About String Methods?



String Methods

Strings implement all of the common sequence operations, along with the additional methods described below.

Strings also support two styles of string formatting, on
tomization (see `str.format()`, Format String Syntax
based on C `printf` style formatting that handles a nar
correctly, but is often faster for the cases it can handle (printf-style String Formatting).

The Text Processing Services section of the standard library covers a number of other modules that provide various text related utilities (including regular expression support in the `re` module).

`str.`**`capitalize`**`()`

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

`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.

In the documentation!

String Processing

A Word Problem

- Suppose you are given a variable s
  - You are not told what is inside of it
  - You only know that it is a string
- Told to find the middle third of string
  - You can only use function and methods
  - Again, no idea what is inside of the string
- What you do has to work for **any** string
  - `s = 'abc'`, answer `'b'`
  - `s = 'abcdef'`, answer is `'cd'`

Implement this Function

```
def middle(text):
    """Returns: middle 3rd of text
    Position, size rounded down
    Precondition: text is a string"""
```

Fill this in

String Processing

- Functions that
  - Take string as argument
  - Produce some value
- 1st *interesting* functions
  - Focus of Assignment 1

What Can We Do With Strings

- We can **slice** strings (`s[a:b]`)
- We can **glue** together strings (+)
- We can use string **methods**
  - We can **search** for characters
  - We can **count** the number of characters
  - We can **pad** strings
  - We can **strip** padding
- Sometimes, we can **cast** to a new type

What Can We Do With Strings

- We can **slice** strings (`s[a:b]`)
- We can **glue** together strings (+)
- We can us
  - We can
  - We can acters
  - We can **pad** strings
  - We can **strip** padding
- Sometimes, we can **cast** to a new type

These will be our
building blocks

Getting Started

- The first step is always the hardest
  - Most students unsure of where to start
  - Will have another video series on this
- **Idea**: Why not work in *reverse*?
  - Specification tells you what to return
  - Figure the operation you need to get there
  - Make a variable if unsure about a step
  - Assign that variable on previous line

Example: Getting the Middle Third

```
def middle(text):
    """Returns: middle 3rd of text
    Position and size are rounded down
    Precondition: text is a string"""






    # Return the final answer
    return result
```

Example: Getting the Middle Third

```python
def middle(text):
    """Returns: middle 3rd of text
    Position and size are rounded down
    Precondition: text is a string"""




    # Cut out the final answer
    result = text[start:end]
    return result
```

Example: Getting the Middle Third

```
def middle(text):
    """Returns: middle 3rd of text
    Position and size are rounded down
    Precondition: text is a string"""



    # Get the end of the middle third
    end = 2*size//3
    result = text[start:end]
    return result
```

Example: Getting the Middle Third

```
def middle(text):
    """Returns: middle 3rd of text
    Position and size are rounded down
    Precondition: text is a string"""

    # Get the start of the middle third
    start = size//3
    end = 2*size//3
    result = text[start:end]
    return result
```

Example: Getting the Middle Third

```
def middle(text):
    """Returns: middle 3rd of text
    Position and size are rounded down
    Precondition: text is a string"""
    # Get the size of the text
    size = len(text)
    start = size//3
    end = 2*size//3
    result = text[start:end]
    return result
```

Testing the Result

```
def middle(text):
    """Returns: middle 3rd of text
    Precond: text is a string"""

    # Get length of text
    size = len(text)
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

```
>>> middle('abc')
'b'
>>> middle('aabbcc')
'bb'
>>> middle('aaabbbccc')
'bbb'
```